TURING 图灵程序设计丛书 Web开发系列

附赠光盘

深入浅出Ext JS

第2版

徐会生 康爱媛 何启伟 著



- 畅销书全新升级，涉及Ext JS 3.2新特性
- Ext JS专家力作，示例丰富，理论和实践并重
- Ajax中国、DOJO中国、开源中国社区、一起Ext四大网站联袂推荐

人民邮电出版社
POSTS & TELECOM PRESS

深入浅出Ext JS 第2版

以用户为中心的时代，应用的界面外观变得越来越重要。然而，很多程序员都缺乏美术功底，要开发出界面美观的应用实属不易。Ext JS的出现，为广大程序员解决了这一难题。它有丰富多彩的界面和强大的功能，是开发具有炫丽外观的RIA应用的最佳选择。

本书是《深入浅出Ext JS》的升级版，涵盖了最新发布的Ext JS 3.2新特性，并对上一版的内容进行增补，充实了示例代码，同时补充了两个功能强大的实例。特别是新增了如何优化基于EXT的应用，提升加载速度，如何创建用户扩展组件以及常用的第三方扩展件等内容。大家可以看到如何在EXT中使用漂亮的图表，尽情欣赏EXT在性能方面实现的巨大突破，以及各种各样的绚丽组件。

本书注重理论与实践相结合，适合各层次Web开发人员阅读。



本书提供配套光盘，内含书中所有示例的完整源代码，几乎所有示例都可以在本地直接运行。此外，光盘还提供了jBPM 4视频教程，供感兴趣的读者学习参考。

徐会生

高级软件工程师兼系统架构师，资深Java EE开发专家。他是国内探索Ext JS的先驱之一，精通UI开发，业余时间为Family168（www.family168.com）撰写了大量开源方面的教程。

康爱媛

从事金融行业软件开发多年，目前是上海一家大型IT公司的高级工程师，利用Java EE和Ext JS为众多行业开发过企业级的系统框架。

何启伟

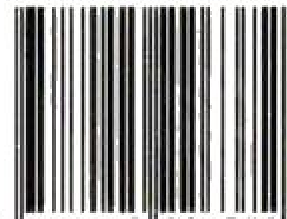
拥有10余年Java EE开发经验，为银行、医疗、烟草等各大行业成功开发了大量企业级应用。他用Ext JS和J2EE开发了一个强大的企业级应用框架——Ext Framework，在国内颇具影响力。同时，他还独立开发了大量EXT扩展组件，深受广大开发者欢迎。

图灵网站：www.turingbook.com　热线：(010)51095186
反馈/投稿/推荐信箱：contact@turingbook.com
有奖勘误：debug@turingbook.com

分类建议　计算机/网络开发/程序设计

人民邮电出版社网址：www.ptpress.com.cn





ISBN 978-7-115-22637-2

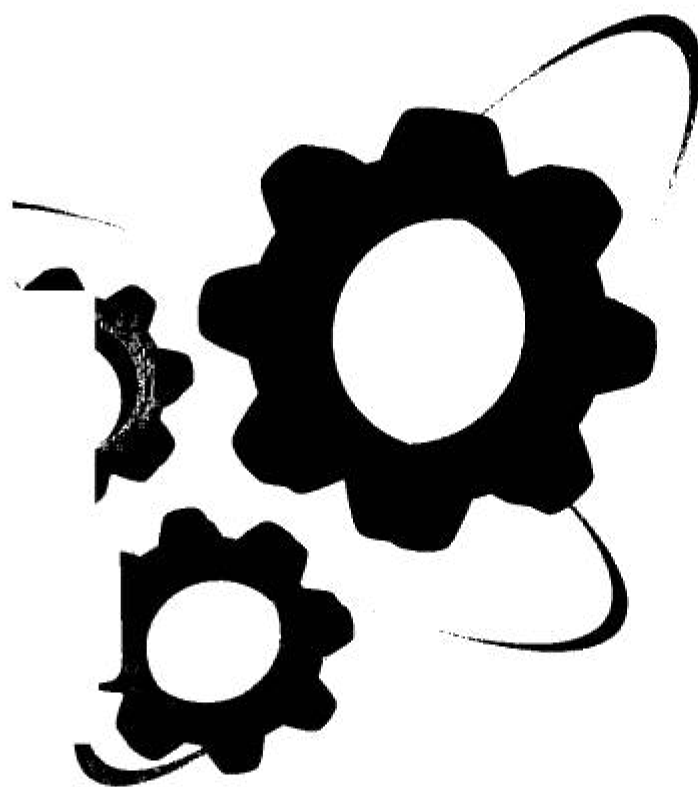
定价：69.00 元（附光盘）

TURING 图灵程序设计丛书 Web开发系列

深入浅出Ext JS

第2版

徐会生 康爱媛 何启伟 著



人民邮电出版社
北京

图书在版编目（CIP）数据

深入浅出Ext JS / 徐会生，康爱媛，何启伟著. --
2版. -- 北京 : 人民邮电出版社，2010.5
（图灵程序设计丛书）
ISBN 978-7-115-22637-2

Ⅰ. ①深… Ⅱ. ①徐… ②康… ③何… Ⅲ. ①
JAVA语言—主页制作—程序设计 Ⅳ. ①TP393.092

中国版本图书馆CIP数据核字(2010)第056711号

内 容 提 要

本书是《深入浅出 Ext JS》的第 2 版，涵盖了 EXT 3.*x* 的新特性，并对上一版进行增补。书中详细讲述了 EXT 的事件、核心组件、表格和表单等各种控件、树形结构、拖放、弹出窗口、布局、数据存储和传输、实用工具等内容，每个知识点都配有相应的示例，可操作性极强，同时补充了两个功能强大的实例，并加入了如何优化基于 EXT 的应用，提升加载速度，如何创建用户扩展组件以及常用的第三方扩展件等内容。

本书适合所有 Web 开发人员阅读。

图灵程序设计丛书

深入浅出Ext JS（第2版）

◆ 著　　徐会生　康爱媛　何启伟
责任编辑　傅志红
执行编辑　杨　爽

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
中国铁道出版社印刷厂印刷

◆ 开本：800×1000　1/16
印张：31.25
字数：738千字　2010年5月第2版
印数：8 801 – 11 800册　2010年5月北京第1次印刷

ISBN 978-7-115-22637-2

定价：69.00 元（附光盘）

读者服务热线：(010)51095186　印装质量热线：(010)67129223

反盗版热线：(010)67171154

前　　言

再次拿起书稿，距离本书第一版的出版已是一年有余，很高兴看到EXT依然在不断向前发展，国内的EXT开发者也越来越多。这些事实都证明了RIA的活力和前景，无论是最终客户还是开发者都期待在B/S结构中获得更加灵活而又强大的功能。这一年中，我们接触、实践、改造也放弃了许多基于EXT的应用系统，这些经验让我们对EXT的原理和应用场景都有了更多的认识和感触。我们也希望将这些积累的经验与大家共享，共同提高。

这次改版的主要目的是将书中的内容从EXT 2.*x*版本升级到EXT 3.*x*，并对上一版中的内容进行增补。实际上，EXT 2.*x*至3.*x*的版本升级并没有为我们带来阵痛，绝大部分EXT 2.*x*中的功能依然可以在EXT 3.*x*中使用，这对开发者来说是一个很好的消息，意味着系统升级会简单许多，我们只需要考虑是否为原有系统添加EXT 3.*x*版本中新加入的特性就可以了，原有的功能依然可以正常使用。

这对我们也是一个好消息，在书的改版过程中，可以把精力更多集中在对新功能的介绍上。我们对这一版新增的内容充满了信心，无论是对EXT 3.*x*中新增功能的介绍，还是在原有内容基础上进行的扩充都闪烁着耀眼的光芒。大家可以看到如何在EXT中使用漂亮的图表，可以尽情感叹EXT在性能方面实现的巨大突破，还有各种各样的绚丽组件，先不管它们好用不好用，只看到它们的显示效果就足够我们惊叹了。

不得不提的是，随着企业对EXT的应用规模逐渐变大，如何对原有功能进行扩展，如何编写自定义组件，如何使用插件等问题开始慢慢浮出水面。我们很高兴看到更多开发者加入到定制EXT组件的行列中，希望这一版中对用户扩展和插件的介绍及实例能够帮助大家一睹EXT定制组件的风采。

任何想学习EXT的开发者都可以通过这本书快速入门。书中包含的所有功能都配有实例，大部分实例都可以直接在本地使用浏览器打开。对于一些必须使用后台脚本支持的实例，我们也尽量使用最简单的后台脚本实现，避免初学者被复杂的配置和框架应用混淆了视听。

大多数实例都可以在本地直接运行，打开随书光盘中的ext-3.0.0/examples文件夹，可以看到书中每一章内容都对应了一个单独的文件夹（如第3章对应的文件夹为03.grid），打开对应的文件夹，双击其中的HTML文件就可以使用浏览器运行实例[1]。

对于那些需要后台脚本支持的实例，需要先确保本机安装了JDK并正确配置了环境变量，然

① 光盘中有一个名为howto.swf的文件，其中以视频的方式讲解了如何查找书中对应的实例以及如何运行这些实例。

后进入随书光盘中的apache-tomcat-5.5.28目录下，执行bin目录下的startup.bat批处理文件，当服务器启动完成后即可使用浏览器访问http://localhost:8080/ext-3.0.0/examples下的实例了。

EXT与AIR的相关实例都放在随书光盘的air目录下，在运行这些实例之前要在本机安装AdobeAIRSDK，然后执行实例中的脚本即可运行实例。

光盘中除了书中所述功能的实例之外，我们还额外附加了一些工作流引擎（jBPM 4.*x*）的演示视频。jBPM 4是family168下一步的发展方向，我们在jBPM 4.*x*之上开发的应用也都使用EXT来实现前台的布局与展现，希望为大家拓展视野。

最后，我们还要感谢人民邮电出版社图灵公司对我们的支持，因为有他们的支持和帮助，我们才能顺利完成本书的改版工作。

希望大家能够喜欢《深入浅出Ext JS》（第2版）。

徐会生　康爱媛　何启伟

2010年1月

第1版前言

Ext JS通常简称为EXT，它是一个非常优秀的Ajax框架，用JavaScript编写，与后台技术无关，可以用来开发具有炫丽外观的富客户端应用。EXT所开发的多彩界面吸引了许多程序员的眼球，同时也吸引了众多客户，它似乎一夜之间就迅速流行开来。对于企业应用系统，尤其是MIS类型的系统而言，EXT非常适用。

当我们第一次使用EXT时，就被它深深地吸引住了。对于我们这些没有美术功底的程序员来说，EXT为我们解决了一大难题，因为它天生拥有炫丽的外表。同时，有很多用其他技术无法实现或极难实现的功能，却能用EXT轻易实现，比如EXT中的表格、树形、布局等控件能为我们的日常开发工作节约大量的时间和精力，这些都坚定了我们使用EXT的决心。

我们在学习EXT的过程中做了大量笔记，记下了学习过程中的一些心得和体会，同时也写了很多示例程序，但是从未想过会将这些资料付诸出版。EXT的参考资料很缺乏，我们发现身边很多学习EXT的朋友都在黑暗中摸索，尤其是英文不太好的朋友，学习起来非常吃力。EXT的中文资料就更少了，虽然有人把EXT官方的API文档中文化了，但是API文档中只有一些基础理论和简单示例，并不能指导我们快速地去实践。我们是实用主义者，本书的最大特点就是以实例为基础，在实例的基础上讲解EXT的各种用法。这样既便于读者理解，也方便读者亲自实践，从而迅速地将所学到的知识运用到实际项目中。

本书适合有一定CSS和HTML基础的开发者阅读，它的主要目的是让开发者能快速学会EXT，并立即付诸实践。书中的示例代码都是以EXT 2.2为基础的，也包含了即将发布的EXT 3.0中的新特性，对EXT的相关知识进行了深入而全面的阐述。

EXT发布包中有一个examples目录，是专门用来放置各种演示示例的，本书附带的所有示例[①]也可以直接放在这个目录下使用。使用时，请将对应目录放在EXT发布包的examples目录下，可以用浏览器打开示例HTML文件观看效果（见图0-1）。

示例中使用了localXHR.js，无需服务器就可以读取JSON数据，从而可以直接在本地浏览大部分示例。对于那些需要后台JSP提供数据的示例，最简单的方法是将整个EXT发布包复制到Tomcat的webapps目录下，启动Tomcat后，可通过浏览器访问examples下的示例。



图0-1　examples目录展开图

① 这些示例可以从图灵公司网站本书配套页面下载。——编者注

出于对EXT的喜爱，又承蒙广大爱好EXT的朋友的厚爱，我们写作了本书。如果有不恰当之处，敬请批评指正。为了便于与读者朋友交流，我们特在我们的网站www.family168.com上为本书开辟了专门的页面。欢迎大家把对本书的意见和建议发表在这个页面上来，我们会积极参与讨论，在此深表感谢。

最后，我们要感谢所有在本书的写作期间给予我们帮助和鼓励的朋友们，还有那些志同道合的EXT爱好者们。

徐会生　何启伟　康爱媛

2008年10月20日

目 录

第 1 章 EXT 概述 …… 1

1.1 EXT 版本变迁 …… 1

1.2 下载 EXT 发布包 …… 2

1.3 如何查看 EXT 自带的 API 和示例 …… 2

1.4 为什么有些示例必须放在服务器上才能看到效果 …… 3

1.5 Hello World …… 3

1.5.1 直接使用下载的发布包 …… 3

1.5.2 在项目中使用 EXT …… 4

1.6 为什么页面提示“找不到图片” …… 5

1.7 辅助开发 …… 5

1.7.1 调试工具 Firebug …… 5

1.7.2 开发利器 Spket …… 8

1.8 小结 …… 12

第 2 章 EXT 框架基础 …… 13

2.1 EXT 的事件和类 …… 13

2.1.1 自定义事件 …… 13

2.1.2 浏览器事件 …… 15

2.1.3 Ext.lib.Event …… 15

2.1.4 Ext.util.Observable …… 16

2.1.5 Ext.EventManager …… 19

2.1.6 Ext.EventObject …… 21

2.2 EXT 的核心组件 …… 22

2.2.1 Ext.Component …… 22

2.2.2 Ext.BoxComponent …… 25

2.2.3 Ext.Container …… 25

2.2.4 Ext.Panel …… 26

2.2.5 Ext.TabPanel …… 26

2.3 小结 …… 29

第 3 章 表格控件 …… 30

3.1 表格的特性简介 …… 30

3.2 制作一个简单的表格 …… 31

3.3 表格常用功能详解 …… 33

3.3.1 部分属性功能 …… 34

3.3.2 自主决定每列的宽度 …… 35

3.3.3 让表格支持按列排序 …… 37

3.3.4 解决中文排序 …… 38

3.3.5 显示日期类型数据 …… 40

3.4 表格渲染 …… 41

3.5 给表格的行和列设置颜色 …… 45

3.6 自动显示行号和复选框 …… 46

3.6.1 自动显示行号 …… 46

3.6.2 复选框 …… 48

3.7 选择模型 …… 49

3.8 表格视图——Ext.grid.GridView …… 50

3.9 表格分页 …… 52

3.9.1 为表格添加分页工具条 …… 52

3.9.2 通过后台脚本获得分页数据 …… 53

3.9.3 分页工具条显示在表格的顶部 …… 57

3.9.4 让 EXT 支持前台分页 …… 57

3.10 后台排序 …… 58

3.11 可编辑表格控件——EditorGrid …… 60

3.11.1 制作一个简单的 EditorGrid …… 60

3.11.2 添加一行数据 …… 62

3.11.3 保存修改结果 …… 65

3.11.4 验证 EditGrid 中的数据 …… 68

3.11.5 限制输入数据的类型 …… 72

3.12 属性表格控件——PropertyGrid …… 76

3.12.1 PropertyGrid …… 77

3.12.2 只能看不能动的 PropertyGrid …… 78
3.12.3 强制对 name 列排序 …… 78
3.12.4 根据 name 获得 value …… 79
3.12.5 自定义编辑器 …… 79
3.13 分组表格控件——Group …… 79
3.13.1 分组表格简介 …… 80
3.13.2 分组表格视图 Ext.grid.GroupingView …… 81
3.14 可拖放的表格 …… 83
3.14.1 拖放改变表格的大小 …… 83
3.14.2 在同一个表格里拖放 …… 85
3.14.3 表格之间的拖放 …… 87
3.14.4 表格与树之间的拖放 …… 90
3.15 表格与右键菜单 …… 91
3.16 小结 …… 93

第 4 章 表单与输入控件 …… 94

4.1 制作表单 …… 94
4.2 FormPanel 和 BasicForm 详解 …… 95
4.3 EXT 支持的控件 …… 95
4.3.1 控件继承图 …… 95
4.3.2 表单控件 …… 96
4.3.3 基本输入控件 Ext.form.Field …… 99
4.3.4 文本输入控件 Ext.form.TextField …… 100
4.3.5 多行文本输入控件 Ext.form.TextArea …… 101
4.3.6 日期输入控件 Ext.form.DateField …… 101
4.3.7 时间输入控件 Ext.form.TimeField …… 102
4.3.8 在线编辑器 Ext.form.HtmlEditor …… 102
4.3.9 隐藏域 Ext.form.Hidden …… 103
4.3.10 下拉输入框 Ext.form.TriggerField …… 103
4.4 使用表单提交数据 …… 105
4.4.1 EXT 默认的提交形式 …… 105
4.4.2 使用 HTML 原始的提交形式 …… 107
4.4.3 单纯 Ajax …… 107
4.5 数据校验 …… 108
4.5.1 输入不能为空 …… 108
4.5.2 最大长度和最小长度 …… 109
4.5.3 借助 vtype …… 109
4.5.4 自定义校验规则 …… 110
4.5.5 算不上校验的 NumberField …… 110
4.5.6 使用后台返回的校验信息 …… 111
4.6 表单布局 …… 112
4.6.1 默认的平铺布局 …… 113
4.6.2 平行分列布局 …… 113
4.6.3 在布局中使用 fieldset …… 116
4.6.4 在 fieldset 中使用布局 …… 118
4.6.5 自定义布局 …… 119
4.7 ComboBox 详解 …… 121
4.7.1 ComboBox 简介 …… 122
4.7.2 将 Select 转换成 ComboBox …… 123
4.7.3 ComboBox 结构详解 …… 123
4.7.4 ComboBox 读取远程数据 …… 125
4.7.5 ComboBox 的高级配置 …… 126
4.7.6 监听用户选择的数据 …… 128
4.7.7 使用本地数据实现省、市、县级联 …… 129
4.7.8 使用后台数据实现省、市、县级联 …… 132
4.8 复选框和单选框 …… 135
4.8.1 复选框 …… 135
4.8.2 单选框 …… 136
4.9 文件上传 …… 137
4.10 自动把数据填充到表单中 …… 138
4.11 小结 …… 140

第 5 章 树形结构 …… 141

5.1 TreePanel 的基本使用 …… 141
5.1.1 创建一棵树 …… 141
5.1.2 为树生枝展叶 …… 142
5.1.3 树形的配置 …… 143
5.1.4 使用 TreeLoader 获得数据 …… 144
5.1.5 读取本地 JSON 数据 …… 145
5.1.6 与 Struts 2 进行集成 …… 146

5.1.7 使用 JSP 提供后台数据 ············ 147
5.2 树的事件 ············ 150
5.3 右键菜单 ············ 151
5.4 修改节点的默认图标 ············ 153
5.5 从节点弹出对话框 ············ 153
5.6 节点提示信息 ············ 154
5.7 为节点设置超链接 ············ 155
5.8 直接修改树节点名称 ············ 156
5.9 树形的拖放 ············ 157
5.9.1 节点拖放的 3 种形式 ············ 158
5.9.2 叶子不能 append ············ 158
5.9.3 判断拖放的目标 ············ 159
5.9.4 树之间的拖放 ············ 161
5.10 树形过滤器 TreeFilter ············ 161
5.11 利用 TreeSorter 对树进行排序 ············ 164
5.12 树形节点视图——Ext.tree.TreeNodeUI ············ 164
5.13 表格与树形的结合——Ext.ux.tree.ColumnTree ············ 166
5.14 小结 ············ 168

第 6 章 拖放 ············ 169
6.1 拖放简介 ············ 169
6.2 拖放的简单应用 ············ 169
6.3 拖放组件体系 ············ 170
6.4 拖放的事件 ············ 172
6.5 高级拖放 ············ 174
6.5.1 基础 ············ 174
6.5.2 控制柄 ············ 174
6.5.3 总在最上面 ············ 175
6.5.4 代理 ············ 177
6.5.5 分组 ············ 178
6.5.6 网格 ············ 182
6.5.7 拖动圆形 ············ 183
6.5.8 拖动范围 ············ 185
6.6 小结 ············ 186

第 7 章 弹出窗口 ············ 187
7.1 Ext.MessageBox ············ 187
7.1.1 Ext.MessageBox.alert() ············ 187
7.1.2 Ext.MessageBox.confirm() ············ 188
7.1.3 Ext.MessageBox.prompt() ············ 188
7.2 对话框的更多配置 ············ 189
7.2.1 可以输入多行的输入框 ············ 189
7.2.2 自定义对话框的按钮 ············ 189
7.2.3 进度条 ············ 190
7.2.4 动画效果 ············ 191
7.3 Ext.window 的常用属性 ············ 191
7.3.1 创建窗口 ············ 192
7.3.2 窗口的最大化和最小化 ············ 193
7.3.3 窗口的隐藏与销毁 ············ 194
7.3.4 防止窗口超出浏览器 ············ 195
7.3.5 设置窗口中的按钮 ············ 196
7.3.6 窗口的其他配置选项 ············ 197
7.4 窗口分组 ············ 198
7.5 向窗口中放入各种控件 ············ 200
7.5.1 在窗口中加入表格 ············ 200
7.5.2 在窗口中加入表单 ············ 201
7.5.3 复杂布局 ············ 202
7.6 小结 ············ 204

第 8 章 布局 ············ 205
8.1 布局的用途 ············ 205
8.2 最简单的布局——FitLayout ············ 207
8.3 常用的边框布局——BorderLayout ············ 209
8.3.1 设置子区域的大小 ············ 210
8.3.2 使用 split 并限制它的范围 ············ 212
8.3.3 子区域的展开和折叠 ············ 213
8.4 制作伸缩菜单的布局——Accordion ············ 216
8.5 实现操作向导的布局——CardLayout ············ 217
8.6 控制位置和大小的布局——AnchorLayout 和 AbsoluteLayout ············ 219
8.7 表单专用的布局 FormLayout ············ 223
8.8 分列式的布局 ColumnLayout ············ 225
8.9 表格状的布局 TableLayout ············ 227
8.10 与布局相关的其他知识 ············ 228
8.10.1 超类 Ext.Container 的公共配置与 xtype 的概念 ············ 228
8.10.2 layout 的超类 Ext.layout.ContainerLayout ············ 229

8.10.3 不指定任何布局时会发生的情况 230
8.10.4 使用 Viewport 对整个页面进行布局 231
8.10.5 使用嵌套实现复杂布局 232
8.11 BoxLayout 236
8.12 小结 237
第 9 章 工具条和菜单 238
9.1 简单菜单 238
9.2 向菜单中添加分隔线 239
9.3 多级菜单 240
9.4 高级菜单 242
9.4.1 多选菜单和单选菜单 242
9.4.2 日期菜单 244
9.4.3 颜色菜单 244
9.4.4 在菜单中添加其他组件 245
9.4.5 使用 Ext.menu.MenuMgr 统一管理菜单 246
9.5 工具条组件详解 248
9.5.1 Ext.Toolbar.Button 248
9.5.2 Ext.Toolbar.TextMenu 248
9.5.3 Ext.Toolbar.Spacer 249
9.5.4 Ext.Toolbar.Separator 250
9.5.5 Ext.Toolbar.Fill 250
9.5.6 Ext.SplitButton 251
9.5.7 为工具条添加 HTML 标签 252
9.5.8 为工具条添加输入控件 253
9.6 分页工具条 Ext.PagingToolbar 253
9.6.1 Ext.PagingToolbar 的基本用法 253
9.6.2 向 Ext.PagingToolbar 添加按钮组件 254
9.7 右键弹出菜单 255
9.8 小结 257
第 10 章 数据存储与传输 258
10.1 Ext.data 命名空间下常用组件简介 258
10.2 Ext.data.Connection 258
10.3 Ext.data.Record 260
10.4 Ext.data.Store 261
10.4.1 基本应用 261
10.4.2 对数据进行排序 262
10.4.3 从 store 中获取数据 263
10.4.4 更新 store 中的数据 264
10.4.5 加载及显示数据 265
10.4.6 其他功能 266
10.5 常用 proxy 268
10.5.1 MemoryProxy 268
10.5.2 HttpProxy 268
10.5.3 ScriptTagProxy 268
10.6 常用 Reader 269
10.6.1 ArrayReader 269
10.6.2 JsonReader 270
10.6.3 XmlReader 271
10.7 高级 store 273
10.8 EXT 中的 Ajax 274
10.8.1 最容易看到的 Ext.Ajax 274
10.8.2 Ext.lib.Ajax 是更底层的封装 275
10.9 关于 scope 和 createDelegate() 276
10.10 DWR 与 EXT 整合 277
10.10.1 在 EXT 中直接使用 DWR 277
10.10.2 DWRProxy 279
10.10.3 DWRTreeLoader 280
10.10.4 DWRProxy 和 ComboBox 281
10.11 localXHR 支持本地使用 Ajax 282
10.12 小结 284
第 11 章 实用工具 285
11.1 EXT 提供的常用函数 285
11.1.1 onReady 函数 286
11.1.2 get 函数 286
11.1.3 query 函数和 select 函数 289
11.1.4 encode 函数和 decode 函数 292
11.1.5 extend 函数 294
11.1.6 apply 函数和 applyIf 函数 295
11.1.7 namespace 函数 295
11.1.8 Ext.isEmpty 函数 296
11.1.9 Ext.each 函数 297

11.1.10 Ext.DomQuery······298
11.2 使用 DomHelper 和 Template 动态生成 HTML······301
11.2.1 使用 DomHelper 生成小片段······301
11.2.2 Ext.DomHelper.applyStyles 函数······304
11.2.3 Template（模板）······304
11.2.4 Ext.DomHelper.createTemplate 函数······307
11.2.5 复杂模板 XTemplate······308
11.3 用 Ext.Utils.CSS 切换主题······310
11.4 悬停提示······311
11.4.1 初始化······311
11.4.2 注册提示······312
11.4.3 标签提示······312
11.4.4 全局配置······312
11.4.5 个体配置······313
11.5 使用 Ext.state 保存状态······314
11.6 使用 fx 实现的动画效果······317
11.7 局部更新网页内容······319
11.8 使用 Ext.util.Format 对数据进行格式化······320
11.9 使用 Ext.util.CSS 管理 CSS 样式······321
11.10 使用 Ext.util.ClickRepeater 处理点击事件······322
11.11 使用 Ext.util.DelayedTask 延时执行函数······323
11.12 使用 Ext.util.TaskRunner 执行循环任务······324
11.13 混合型集合 Ext.util.MixedCollection······325
11.14 使用 Ext.util.TextMetrices 获得文本所占的高度和宽度······329
11.15 使用 Ext.KeyNav 处理导航按键······330
11.16 使用 Ext.KeyMap 为对象绑定按键功能······331
11.17 扩展······333
11.17.1 扩展 Date······333
11.17.2 扩展 String······335
11.17.3 扩展 Function······336
11.17.4 扩展 Number······338
11.17.5 扩展 Array······338
11.18 门户组件 Ext.ux.Portal······338
11.19 桌面组件 Ext.Desktop······341
11.20 小结······345
第 12 章 一个完整的 EXT 应用······346
12.1 确定整体布局······347
12.2 使用 HTML 和 CSS 设置静态信息······348
12.3 对学生信息进行数据建模······349
12.4 在页面中显示学生信息列表······353
12.5 添加表单编辑学生信息······358
12.6 为表单添加提交事件······361
12.7 清空表单信息······364
12.8 删除指定的学生信息······364
12.9 在表格和表单之间进行数据交互······365
12.10 提升加载速度······366
12.10.1 对 JavaScript 文件进行压缩混淆······367
12.10.2 使用客户端缓存······367
12.10.3 使用 GZIP 压缩······368
12.11 小结······370
第 13 章 复杂实例······371
13.1 VIP 客户统计系统······371
13.2 Tracker 任务跟踪系统······380
13.3 小结······387
第 14 章 EXT 3.x 中的新特性······388
14.1 介绍 Ext Core······388
14.1.1 adapter······388
14.1.2 core······389
14.1.3 data······389
14.1.4 util······389
14.1.5 扩展实例······390
14.2 介绍 Ext Direct······392
14.2.1 Ext Direct······392
14.2.2 洞悉 Ext Direct 的原理······394
14.2.3 使用 directjngine 支持 Ext Direct······396

14.3 介绍 EXT 3.0 中新增的组件 ······ 398
14.3.1 行编辑器 ······ 399
14.3.2 进度条分页组件 ······ 399
14.3.3 缓冲式表格视图 ······ 400
14.3.4 标签面板的滚动菜单 ······ 401
14.3.5 处理工具条溢出 ······ 401
14.3.6 列表视图 ······ 402
14.3.7 工具条中的分组按钮 ······ 403
14.3.8 高级按钮 ······ 403
14.3.9 竖直分组的标签面板 ······ 404
14.4 在 EXT 3.0 中使用 Flash 报表 ······ 405
14.4.1 柱状图 ······ 405
14.4.2 横向柱状图 ······ 406
14.4.3 折线图 ······ 407
14.4.4 饼状图 ······ 408
14.4.5 柱状栈图 ······ 409
14.4.6 横向柱状栈图 ······ 410
14.4.7 混合图 ······ 411
14.5 EXT 3.1 带来的新特性 ······ 412
14.5.1 解决内存泄露 ······ 413
14.5.2 核心组件优化 ······ 414
14.5.3 分组表头 ······ 414
14.5.4 锁定列 ······ 415
14.5.5 树形表格 ······ 416
14.5.6 竖直布局 ······ 418
14.5.7 高级表格查询 ······ 419
14.5.8 自定义编辑器 ······ 421
14.6 EXT 3.2 带来的新特性 ······ 422
14.6.1 多重排序 ······ 423
14.6.2 为 DataView 添加动画变换效果 ······ 423
14.6.3 组合表单控件 ······ 424
14.6.4 滑动条表单控件 ······ 424
14.6.5 为滑动条指定多个滑块 ······ 425
14.6.6 更多工具条插件 ······ 426
14.6.7 新主题 Accessibility ······ 428
14.7 小结 ······ 428
第 15 章 用户扩展与插件 ······ 429
15.1 介绍用户扩展 ······ 429
15.2 编写用户扩展所需的基础知识 ······ 432
15.2.1 继承模型 ······ 432
15.2.2 了解 Component 的生命周期 ······ 436
15.3 编写自定义用户扩展 ······ 437
15.4 介绍 EXT 的插件体系 ······ 438
15.5 常用扩展组件（一）UploadDialog ······ 441
15.6 常用扩展组件（二）ManagedIFrame ······ 443
15.7 小结 ······ 446
附录 A EXT 常见问题 ······ 447
附录 B EXT 对 AIR 的支持 ······ 454
附录 C EXT 的版本变迁 ······ 462

第 1 章

EXT概述



本章内容

- EXT版本变迁
- 下载EXT发布包
- 如何查看EXT自带的API和示例
- 为什么有些示例必须放在服务器上才能看到效果
- Hello World
- 为什么页面提示"找不到图片"
- 辅助开发

1.1 EXT 版本变迁

在正式介绍EXT的详细功能之前，先让我们了解一下其各个版本的功能变迁，从这些版本变动历史中可以对EXT的进化历程有一个大致的了解。

- EXT 1.0发布于2007年2月，这标志着EXT正式从YUI社区中独立出来，不再仅仅支持YUI，而是提供了ext-baes、prototype、jquery和YUI4种底层实现。这个版本提供了基本的表格、树形、表单、窗口和布局组件。EXT采取多协议发布方式，用户可以选择在LGPL协议和企业协议下使用EXT。
- EXT 2.0 发布于2007年12月，这次大版本升级重写了核心组件部分，简化了组件的布局和配置。EXT 2.0之后我们可以使用`layout`和`items`属性更加方便地实现各种复杂布局，而且EXT 2.0中也提供了许多功能强大的布局方式，比如EXT 1.*x*中的Accordian布局方式，已经成为EXT 2.0默认发布包的一部分了。
- EXT 2.1发布于2008年4月，这次版本升级包括对REST的支持，并提供了一些扩展组件。不过这次版本升级做出的最大改变是对开源协议的修改。自EXT 2.1之后，所有未付费的用户都只能在GPL协议下才能使用EXT，也就是说EXT 2.1以及之后的版本都无法直接用于商业项目，必须向EXT缴费购买商业授权才能在商业产品中使用EXT。
- EXT 2.2发布于2008年8月，这个版本提供了对浏览器Firefox 3.*x*的支持，并提供了多种高级表单输入控件，比如可以实现单选多选框横排的RadioGroup和CheckboxGroup，多选列表MultiSelect和ItemSelector以及文件上传组件FileUploadField。

- EXT 2.2.1发布于2009年2月，这个版本提供了对Chrome浏览器的支持，解决了一些内存泄露问题，并为Container提供了`removeAll()`函数，可以直接清空容器内的所有组件，同时提供了多种AIR下的扩展组件。
- EXT 3.0发布于2009年6月，EXT 3.*x*中最主要的变化就是重构并提炼了Ext Core，同时重写了Button和Toolbar，使EXT 3.*x*中的按钮和工具条都可以实现多种显示方式，如自动填充、图文混排，并对工具条提供了溢出控制。
- EXT 3.1发布于2009年12月，这是目前最新的一个发布版本了，此版本中最大的变化是对性能的提升，并解决了单页面应用长期使用时出现的内存泄露问题。

从上面这些版本变迁历史中可以看出，EXT最初以一个扩展组件库的形式发展起来，然后逐渐独立发展。EXT 1.0发布时已经拥有了比较成熟的各种组件。EXT 1.0至EXT 2.0的版本升级过程中实现了对整体组件的重构，实现了更加简易的配置和布局方式。从EXT 2.0之后，EXT的发展开始进入平稳状态，每次升级都只是提供一些扩展组件和对bug的修改。也是因为EXT 2.0中的组件结构已经相当成熟，所以EXT 2.0到EXT 3.0并没有经历像EXT 1.0到EXT 2.0的巨大动荡，只需要对原有代码进行少量修改就可以实现平滑升级。EXT 2.*x*至EXT 3.*x*的版本升级主要体现在核心组件的重构并提取了Ext Core，而真正振奋人心的还是EXT 3.1版本的发布，因为EXT 3.1中，EXT团队终于开始解决性能和内存泄露方面的问题了，并对外宣称在一些场景下可以实现50倍的性能提升，只此一点就可以让之前抱怨项目性能问题的开发者放心了。

我们从这些变迁历史中可以切实感觉到，EXT已经经历了从加强功能结构成熟性发展到加强整体架构和稳定的过渡，这也说明EXT的功能已经足够强大了，今后它会向更快、更稳定的方面努力。

1.2 下载 EXT 发布包

很高兴可以带领大家走进EXT的世界！EXT是一款富客户端开发框架，它基于JavaScript、HTML和CSS开发而成，无需安装任何插件即可在常用浏览器中创建出绚丽的页面效果。非常幸运的是，我们可以从www.extjs.com/download免费获得EXT发布包，其中源代码、API文档和示例一应俱全。不过，如果想通过访问SVN获得最新的代码，就得花钱了。为学习之用，我们现阶段只需要这个免费的发布包，通过其中的范例，我们可以感受一下EXT的魅力。

EXT的作者Jack Slocum[①]在2.2版本之后对EXT的开源协议进行了修改，而且从3.0版本之后又改成了只对付费用户提供补丁维护版本，因此我们能从www.extjs.com上下载到的免费最新版本是EXT 3.1。

1.3 如何查看 EXT 自带的 API 和示例

EXT发布包中的API文档放在docs目录下，虽然可以看到左边的菜单，但是点击之后，右侧的API页面都是通过Ajax方式获得的，不能直接在本地查看，必须把解压缩后的完整目录部署到

① Jack Slocum，美国人，EXT框架之父，首席软件设计师。个人博客：http://www.jackslocum.com/。

服务器上，然后通过浏览器访问服务器，这样才能看到。如果没有把这些内容放在服务器上，则docs就会打不开，只能看到图1-1的界面。



图1-1　通过Ajax方式无法从本地文件系统获取数据

API文档中的官方示例可以在API Home中找到，也可以在浏览器中直接打开examples目录下的samples.html。当然，有一些示例需要放在服务器上才能看到效果。有一些示例的后台代码是使用PHP编写的，如果想查看这些示例的效果，还需要配置PHP运行环境。

如果你用Java，而且JDK的版本在1.5以上，那么建议你安装resin-3，因为可以直接在它上面运行PHP示例。

1.4　为什么有些示例必须放在服务器上才能看到效果

有些示例用Ajax从后台读取数据，如果该示例不在服务器上，Ajax就会一直返回失败状态，从而无法获得任何数据，所以就看不到正确的效果。不过，在www.extjs.com网站的论坛上曾经有人写了localXHR，可以通过Ajax方式从本地文件系统获得数据，这样也许就可以摆脱服务器的束缚了。随书代码中包含了localXHR.js，直接将此文件复制到你的应用中，即可实现使用Ajax从本地文件系统中直接获得数据。

1.5　Hello World

为初学者考虑，我们提供了两个入门版的Hello World范例。

1.5.1　直接使用下载的发布包

如果你已经从http://www.extjs.com/download下载了EXT的zip格式的发布包，那么可根据如下步骤来使用它。

(1) 将该发布包解压缩，其目录结构应该如图1-2所示，各目录的用途简要介绍如下。

- adapter目录下是EXT的核心代码和底层库，包括jQuery、Prototype和YUI的适配器。
- docs目录下是EXT的文档，其实最主要和最重要的是EXT的API，EXT开发中离不开它。
- examples目录下是官方的演示示例，是初学者学习EXT的最佳途径之一。
- pkgs目录下是EXT压缩后的代码，经过压缩的代码，体积更小，加载更快。
- resources目录下是EXT要用到的图片文件和样式文件，EXT绚丽的外观全部由这个目录中的文件控制。

- src目录下是EXT的源码文件，也就是相对pkgs目录而言，未经过压缩的代码。
- ext-all.js文件是EXT的核心库，是必须引入的。
- ext-all-debug.js文件是ext-all.js的调试版，在调试时使用这个调试版本的文件才能正确定位出现错误的位置。
- INCLUDE_ORDER.txt文件用来说明在页面上引用底层库的JavaScript文件的顺序。
- LICENSE.txt文件是EXT的使用许可文件。

(2) 利用它的目录结构编写一个Hello World示例。

进入图1-2中的examples目录[①]，新建一个helloworld目录，在helloworld目录下新建一个helloworld.html文件，将下面的代码复制到这个文件中。

```
<link rel="stylesheet" type="text/css" href="../../resources/css/ext-all.css" />
<script type="text/javascript" src="../../adapter/ext/ext-base.js"></script>
<script type="text/javascript" src="../../ext-all.js"></script>
<script type="text/javascript" src="../examples.js"></script>
<script>
Ext.onReady(function(){
    Ext.MessageBox.alert('helloworld', 'Hello World.');
});
</script>
```

(3) 双击打开helloworld.html文件，其效果如图1-3所示。



图1-2　EXT发布包解压后的目录结构



图1-3　Hello World示例

很高兴地告诉你，我们的Hello World示例已经正确地执行了。建议你接下来把examples目录下的示例都看一看，并研究一下其中的代码是如何写的，好好感受一下EXT的风格，然后再继续。

1.5.2　在项目中使用 EXT

如果想把EXT应用到项目中，那么需要自己整理一下，因为发布包里的内容并非都是必需的，比如文档、示例和源代码。必需内容至少应包括：ext-all.js、adapter/ext/ext-base.js、src/locale/ext-lang-zh_CN.js和整个resources目录。

① 打开光盘，进入ext-3.0.0下的examples文件夹，本章的实例放在01.overview文件夹下。

- ext-all.js和adapter/ext/ext-base.js已经包含了EXT的所有功能，所有的JavaScript脚本都在这里。
- src/locale/ext-lang-zh_CN.js是简体中文国际化资源文件。
- resources目录下是CSS样式表和图片。

只要自己的项目中包含上述内容，就可以使用EXT了。使用时，在页面中导入下面的代码：

```
<link rel="stylesheet" type="text/css" href="${放置ext的目录}/resources/css/ext-all.css"/>
<script type="text/javascript" src="${放置ext的目录}/ext-base.js"></script>
<script type="text/javascript" src="${放置ext的目录}/ext-all.js"></script>
<script type="text/javascript" src="${放置ext的目录}/ext-lang-zh_CN.js"></script>
```

导入时，请注意JavaScript脚本的顺序。

1.6 为什么页面提示“找不到图片”

EXT里经常用一张空白图片作为占位符号，然后用CSS里配置的背景图片显示，这样有利于更换主题。这张空白图片的名字就是Ext.BLANK_IMAGE_URL，默认情况下它的值是`BLANK_IMAGE_URL : "http:/"+"/extjs.com/s.gif "`。图片虽然很小，但是也需要从网上下载，一旦下载失败，就会显示找不到图片。

看到这里，可能有人会觉得奇怪，为什么examples目录下的示例会出现找不到图片的情况呢？如果你提出这样的问题，证明你没有仔细研究那些示例的代码，每个示例都引用了../shared/examples.js，在这个examples.js的第一行就已经设置了`Ext.BLANK_IMAGE_URL = '../../resources/images/default/s.gif';`。所以，要解决自己写的示例找不到图片的问题，只需要参照examples.js中的方法，修改自己项目中的s.gif的本地路径即可。

1.7 辅助开发

在软件开发中，经常会使用辅助开发工具，因为辅助工具能提高开发效率，甚至可以达到事半功倍的效果。尤其是像JavaScript这样的解释型脚本语言，开发和调试过程都非常困难，需要强有力的工具加以支持。下面将介绍在EXT开发中用得最多的调试工具和IDE。

1.7.1 调试工具 Firebug

由于我们对Firefox的偏爱，以及Firebug在调试JavaScript过程中的便利，推荐你使用Firefox和Firebug进行EXT的开发调试。实际上，EXT开发者也都倾向于使用Firefox进行开发，因为有些应用在Firefox上运行良好，在IE中运行却会出现这样或那样的问题。只是，目前IE占据大约60%的浏览器市场份额，所以我们还是需要让自己的项目能在IE中正常运行，这要求我们能编写出跨浏览器的JavaScript代码。

Firebug的好处在于，它可以显示动态生成的DOM，甚至可以在Firebug里直接对DOM进行修改，而这些修改会反映到显示页面上。

通过Firebug提供的控制台，可以直接执行JavaScript脚本，也可以在JavaScript代码中使用`console.debug()`、`console.info()`和`console.error()`等日志方法，便于调试和跟踪。

Firebug可以查看以Ajax方式发送和接收的各种信息，还可以查看发送的参数以及返回的状态和信息。下面将介绍Firebug的安装和使用。

首先下载Firebug，如果你使用的是Firefox 3.5以上的版本，就必须下载Firebug 1.4以上的版本，否则会不兼容。如图1-4所示，在官方网站点击下载链接，Firefox会自动提示安装插件，等待安装完成，重启Firefox之后就可以使用Firebug了。

安装完成后，在Firefox的工具栏上单击“查看”选项，便会看到Firebug，选中它就可以开始调试了，如图1-5所示。

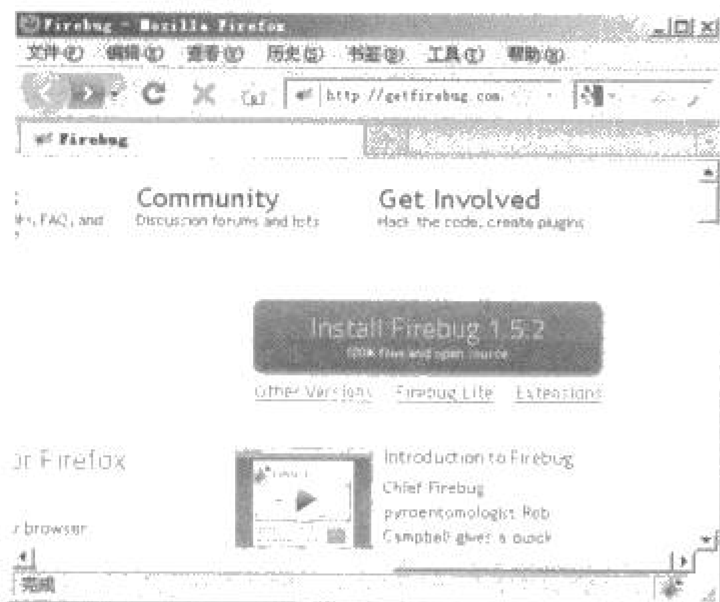


图1-4　在官方网站下载并安装Firebug

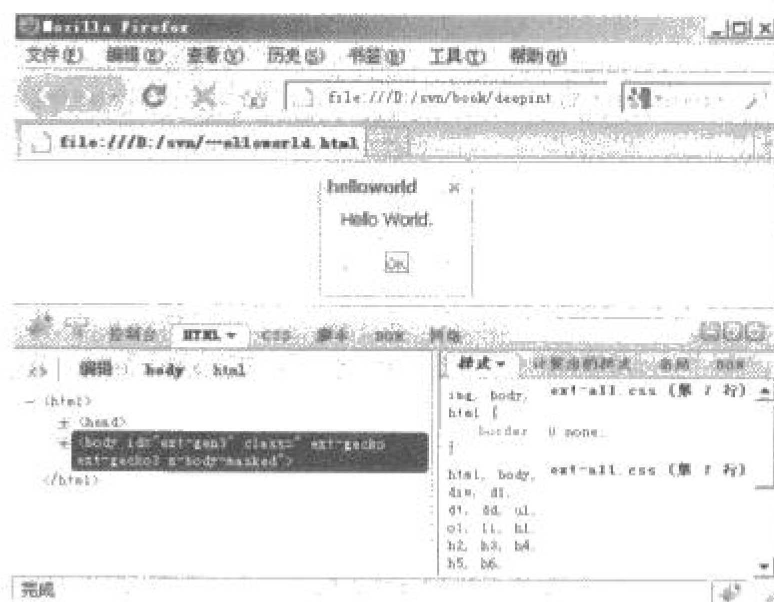


图1-5　Firebug的调试页面

从图1-5中可以看出，Firebug面板的最上方有6个选项卡，点击这些选项卡就可以使用对应的功能操作。在浏览器下方状态栏的右侧可以看到一个昆虫图标，这就是Firebug的图标，当图标颜色为灰色时，表明Firebug处于未启用状态。下面将详细介绍这些选项卡的功能。

- 控制台：控制台会输出代码中的错误，并且可查看错误的详细信息，还可以直接在下方的命令栏中执行JavaScript脚本。同时，右下角Firebug图标的位置也会显示脚本错误的数量，如图1-6所示。

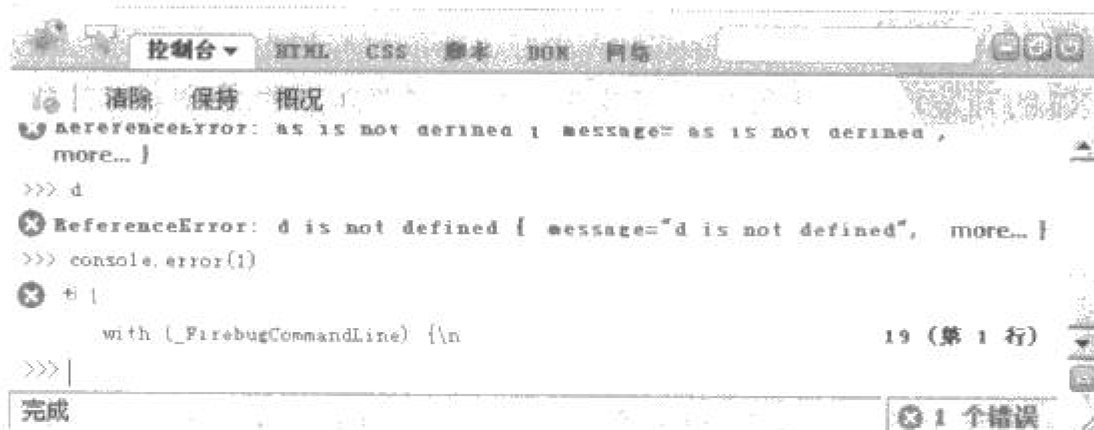


图1-6　控制台输出错误信息

- HTML：HTML查看器。通过它可以查看页面的源代码，这里的源代码包括页面中引用的所有JavaScript脚本和CSS样式代码。也可以单击按钮，再单击选择上方页面的显示区域，将选中的页面元素自动显示在HTML选项卡中。HTML查看器面板的右侧还提供了查看布局的功能，这里提供了可视化的CSS标尺。当在Firebug中查看页面中某一区域的CSS样式表时，它会以标尺的形式将当前区域占用的面积（以像素为单位）清楚地标记出来。而且还能在这个可视化的界面中直接修改各面积的像素值，页面上各区域的位置就会随之变化。当页面中某些元素出现错位时，可以在这里分析外边距、边框、内边距和元素大小之间的关系，如图1-7所示。

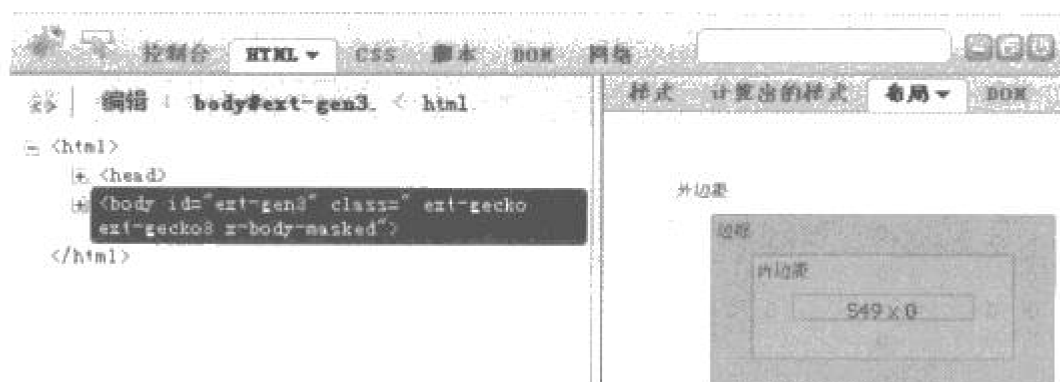


图1-7 查看元素的布局信息

- CSS：CSS调试器。调试CSS样式，通过右侧的样式标签可以查看HTML元素使用的样式。可以在这里直接添加、修改、删除CSS样式表的属性，而且可以在页面中直接看到修改后的结果，如图1-8所示。

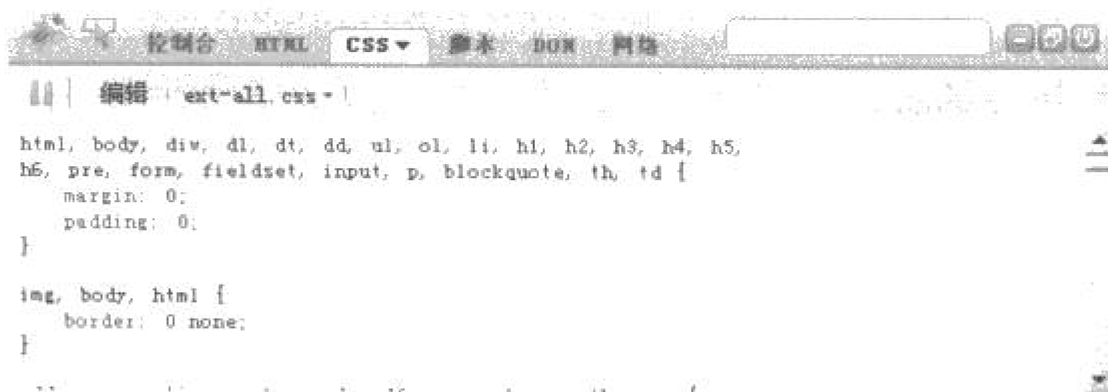


图1-8 CSS调试器

- 脚本：JavaScript脚本调试器。在其中可以为JavaScript脚本设置断点进行单步调试，右侧还提供了查看变量的窗口，如图1-9所示。
- DOM：DOM查看器。查看浏览器中的所有DOM节点，包括由EXT自动创建的组件中的DOM。
- 网络：网络状况监视。它能将页面载入CSS样式文件、JavaScript脚本和网页中引用的外部图片所消耗的时间以柱状图呈现出来，帮助我们找出导致网页访问速度变慢的“元凶”，进而对网页进行优化，这里还可以查看HTTP协议的请求和响应信息，如图1-10所示。

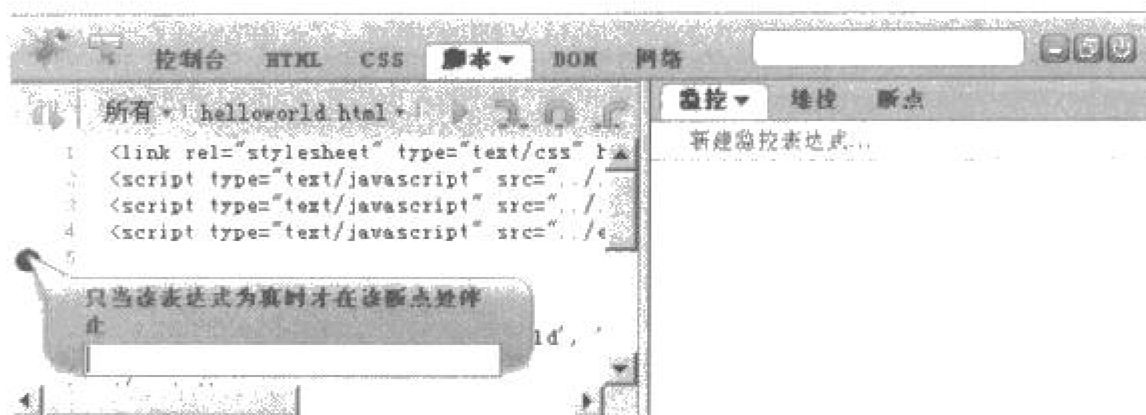


图1-9　JavaScript调试器



图1-10　网络监视器

对Firebug的介绍就讲到此，在使用EXT时，我们会经常使用Firebug来进行开发调试， 如何让Firebug在实际开发中发挥最大的作用，还需要大家在实践中慢慢体会。

1.7.2　开发利器 Spket

Spket是一款全面支持JavaScript、XUL/XBL、Flex、SVG以及Yahoo! Widget的免费开发工具。它体积小巧，只有5.6 MB，目前最新版本是1.6.18。它就像是为EXT量身定做的一样，既可以作为Eclipse的插件，又可作为独立的IDE。下面将介绍Spket的安装和使用。

需要在Spket的官方网站（http://www.spket.com/）下载Eclipse的插件（需要在Eclipse 3.2以上版本中使用）或独立开发IDE，下载文件为spket-1.6.18.zip。

先介绍如何将Spket作为Eclipse的插件来使用。Eclipse插件的安装方法大家应该很熟悉。首先将Spket下载包中eclipse文件夹下的features和plugins文件夹复制到Eclipse安装目录下的eclipse文件夹中；然后启动Eclipse，依次进入Windows→Preferences选项，在窗口的左边可以看到Spket菜单项，其中包含了该插件的相关配置。选中JavaScript Profiles选项，会出现JavaScript配置列表，如图1-11所示。默认情况下，它没有提供对EXT的支持，所以下面我们需要将EXT库添加到列表中。

单击右侧的New按钮，在弹出的窗口中输入“EXT”，单击OK按钮便可以看到EXT的名字已出现在列表中，如图1-12所示。

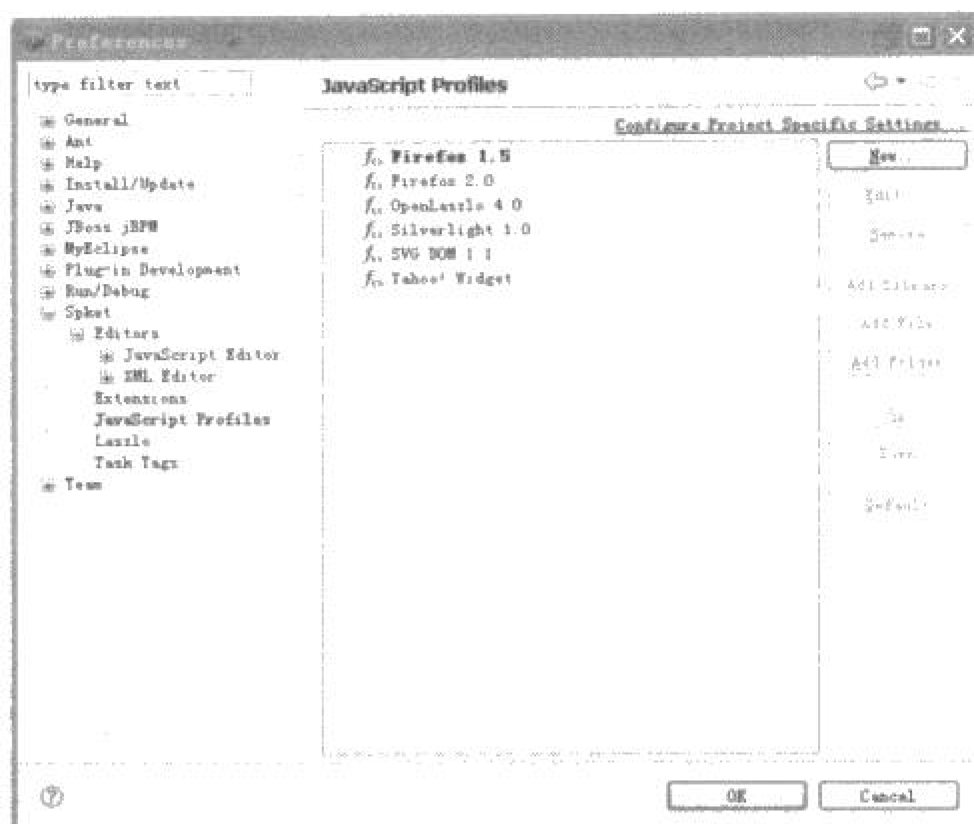


图1-11 Spket插件目录结构

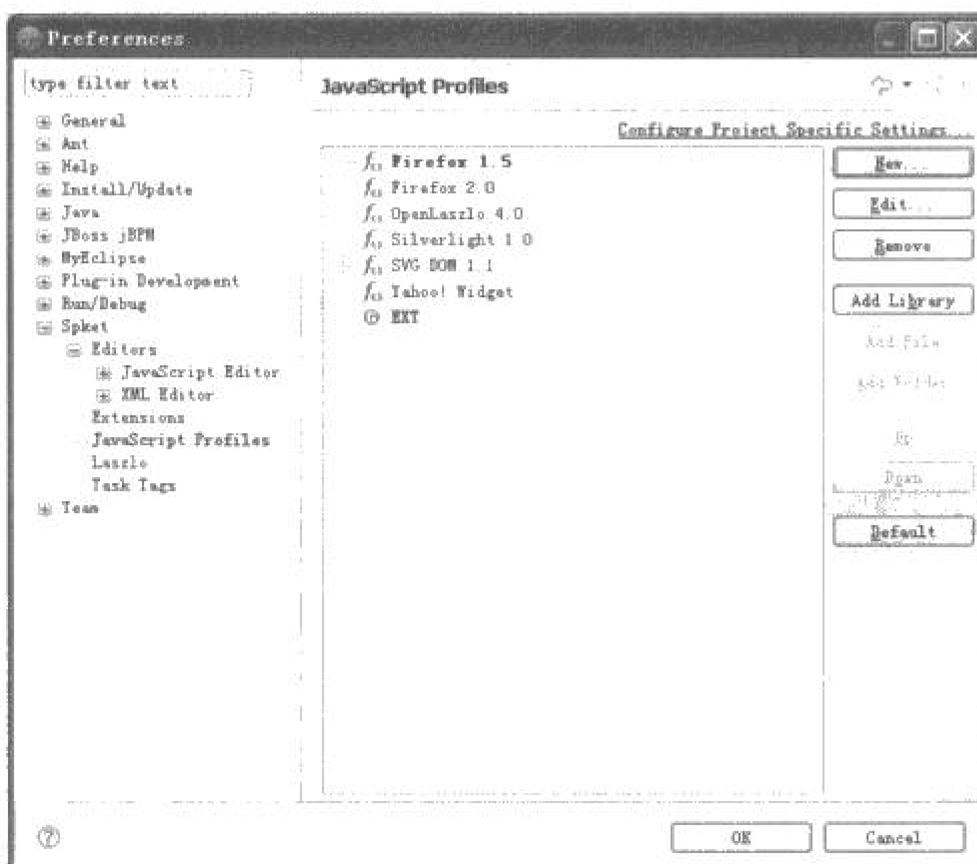


图1-12 JavaScript列表

选择EXT库列表，再单击右侧的Add Library按钮，在弹出的窗口中选择ExtJs，然后单击OK按钮，便会看到EXT库列表下成功添加了ExtJs类库，如图1-13所示。

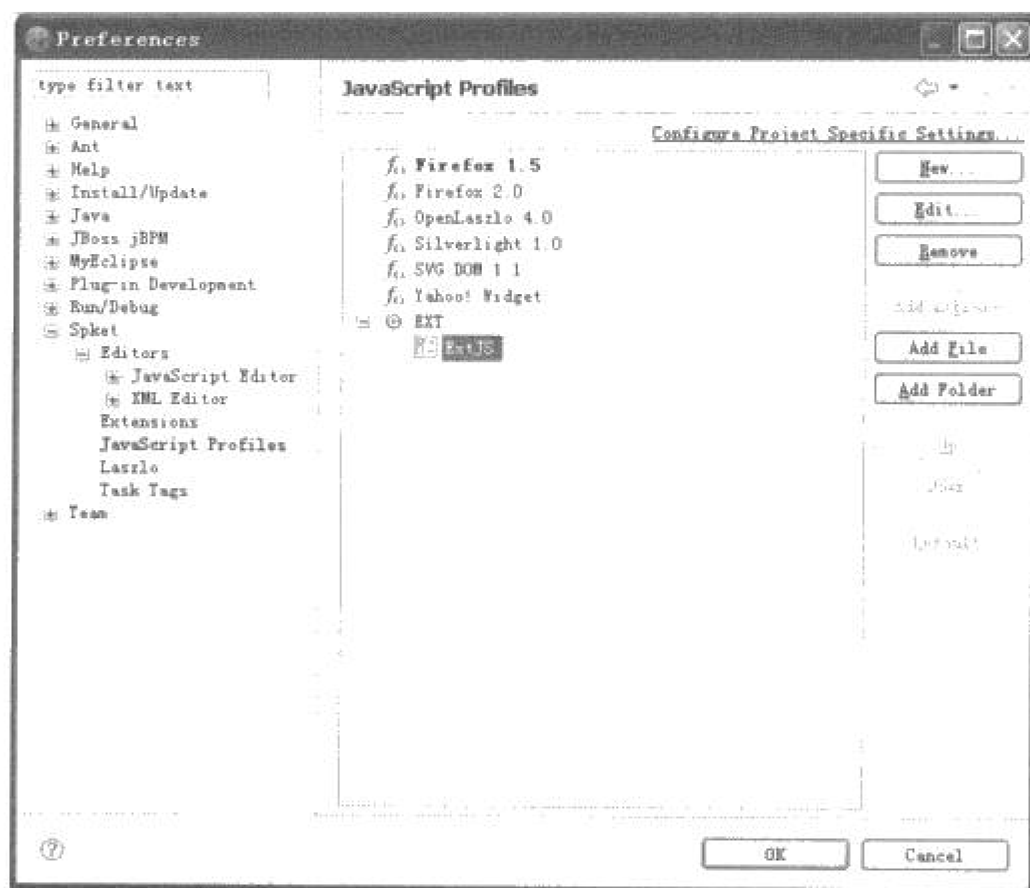


图1-13　ExtJs类库列表

选中ExtJs库列表，单击右侧的Add File按钮，便可以上传库文件，如图1-14所示。

选择ext.jsb文件并打开，可以看到ExtJs类库列表下多了很多文件。默认选择了Ext Base和Everything，Ext Base是EXT的核心库，Everything中包含了EXT的source目录下的所有JavaScript文件，如图1-15所示。

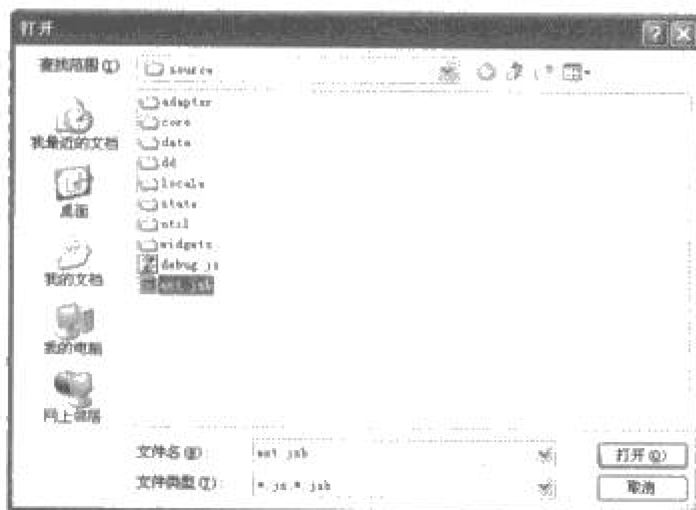


图1-14　选择上传EXT工程文件

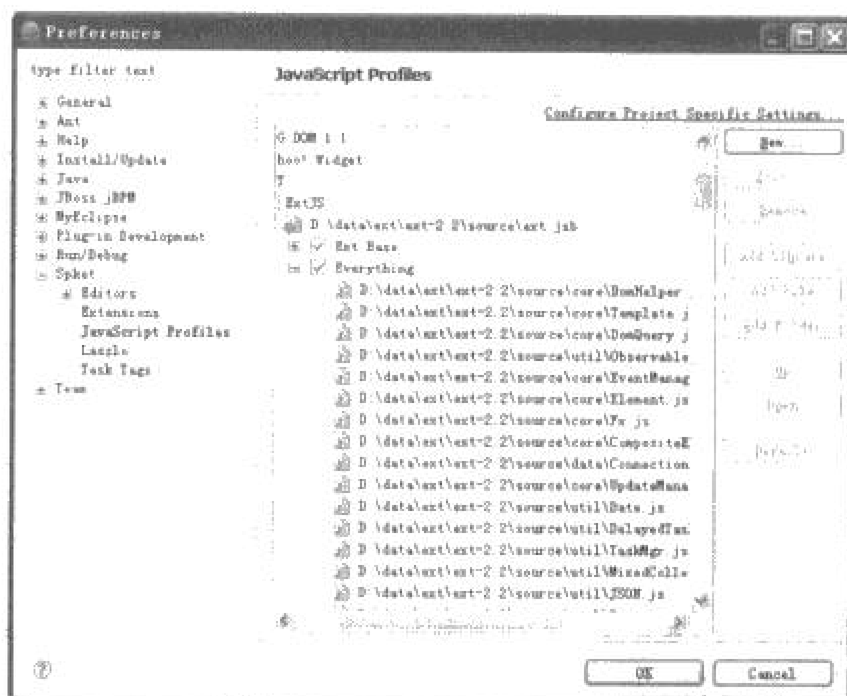


图1-15　ExtJs库文件列表

到这里，Spket插件的安装和配置就已经完成了。下面我们就来体验一下Spket的使用效果，它可以实现EXT的代码提示，如图1-16所示。如果没有提示，需要选中EXT库列表，再单击右侧的Default，这一步是必不可少的。

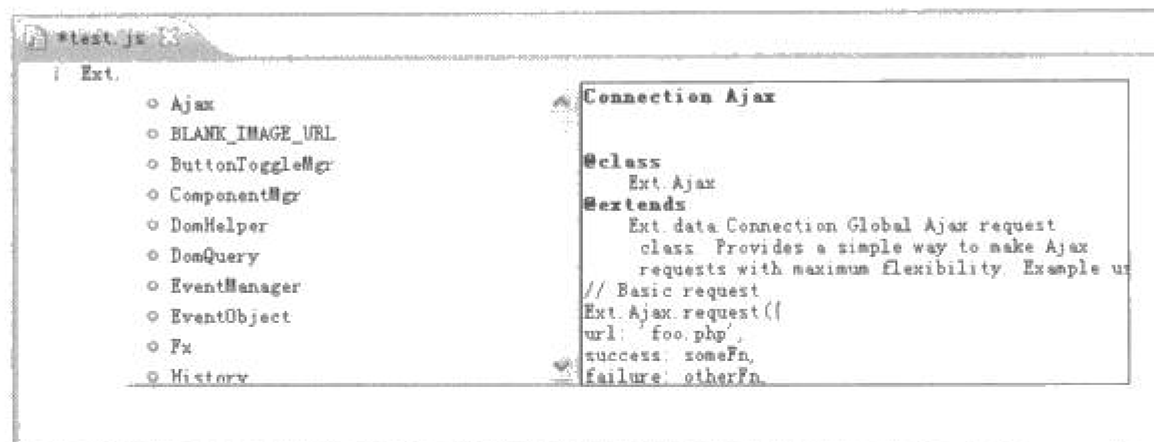


图1-16　Spket插件的代码提示

再介绍一下将Spket作为一个独立IDE的使用方式。作为独立的IDE，Spket需要运行在JDK 1.5版以上，下载文件为spket-1.6.18.jar，安装步骤如下。

(1) 打开操作系统的命令行窗口（cmd控制台）。

(2) 进入spket-1.6.18.jar文件目录，输入java -jar spket-1.6.18.jar。

(3) 回车后便可看到安装画面，这里需要选择将Spket作为Eclipse的插件还是独立作为独立的IDE运行，我们这里选择后者，如图1-17所示。

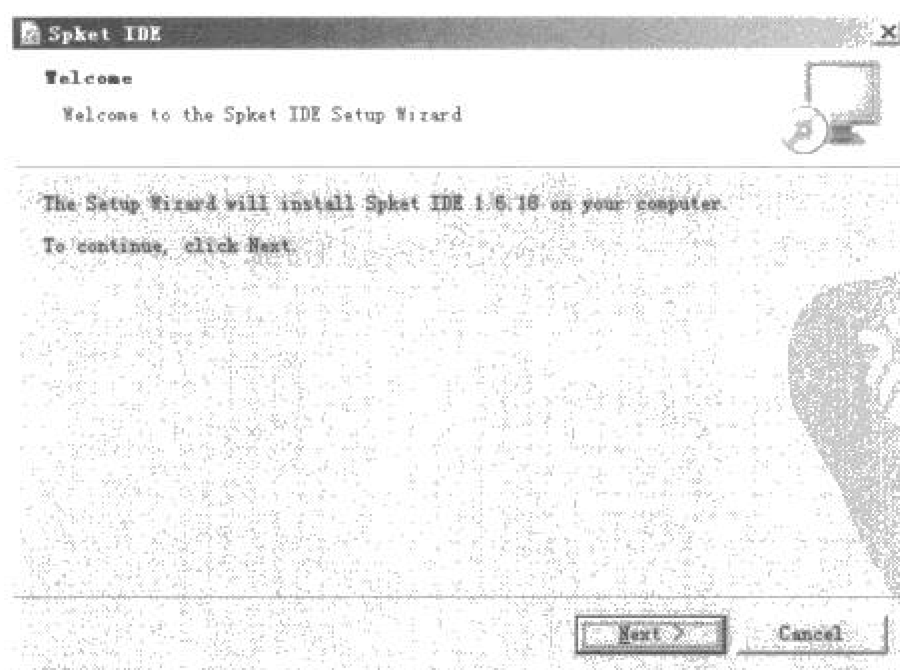


图1-17　安装Spket IDE

然后一步一步往下安装即可，安装完成后会看到一个和Eclipse IDE非常像的界面，使用方式也基本相同，如图1-18所示。

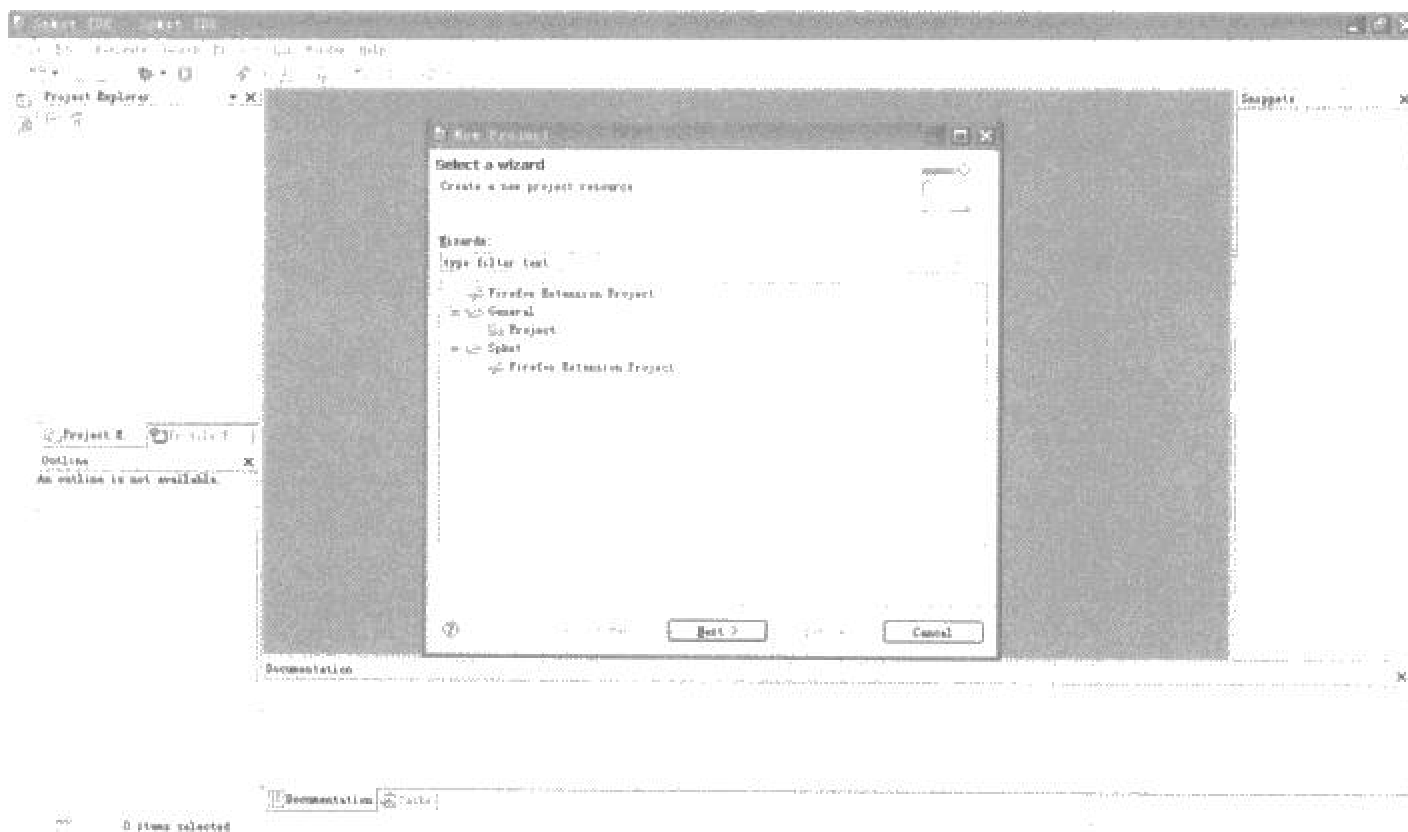


图1-18　Spket IDE的使用界面

将Spket IDE安装完后，我们可以按照之前介绍的在Eclipse插件中使用EXT的方式，对Spket IDE进行配置，这样就可以在Spket中使用EXT进行开发了。

除Spket之外，在EXT开发时用得较多的IDE还有JSEclipse以及Apatana，在此就不一一介绍了。

1.8　小结

本章主要介绍了如何下载和使用EXT发布包，以及查看EXT自带的API和示例，并带领大家制作了第一个EXT示例——Hello World。同时讨论了在最初使用EXT时有可能遇到的一些问题，并介绍了两款常用的辅助开发工具——Firefox的Firebug和Spket。

第 2 章

EXT框架基础

2

本章内容

- EXT的事件和类
- EXT的核心组件

2.1 EXT 的事件和类

事件模型在EXT应用中有着尤为重要的作用。EXT中的事件分为两种类型：自定义事件和浏览器事件。

2.1.1 自定义事件

EXT中遵循一种树状的事件模型，所有继承自Ext.util.Observable类的控件都可以支持事件。可以为这些继承了Ext.util.Observable的对象定义一些事件，然后为这些事件配置监听器。当某个事件被触发时，EXT会自动调用对应的监听器，这些就是EXT的事件模型。

下面通过继承Ext.util.Observable来实现一个支持事件的对象，实现过程如代码清单2-1所示。

代码清单2-1 Person类

```
Person = function(name) {
    this.name = name;
    this.addEvents("walk", "eat", "sleep");
}
Ext.extend(Person, Ext.util.Observable, {
    info: function(event) {
        return this.name + 'is' + event + 'ing.';
    }
});
```

以上代码实现了一个名称为Person的对象，它有一个属性name。初始化时，调用this.addEvents()函数定义了3个事件：walk、eat和sleep，然后使用Ext.extend()让Person继承Ext.util.Observable的所有属性。此外，还加上了函数info()，让它返回Person的信息。

接下来该如何做呢？

我们在HTML中创建一个Person的实例，然后为它的事件配置好监听器，如代码清单2-2所示。

代码清单2-2　为person添加事件监听器

```
var person = new Person('Lingo');
person.on('walk', function() {
    Ext.Msg.alert('event', person.name + "在走啊走啊。");
});
person.on('eat', function(breakfast, lunch, supper) {
    Ext.Msg.alert('event', person.name+"要吃"+breakfast+", "+lunch+"和"+supper+"。");
});
person.on('sleep', function(time) {
    Ext.Msg.alert('event', person.name + "从" + time.format("H") + "点开始睡觉啦。");
});
```

这里的on()是addListener()的简写形式，功能完全一样。第一个参数传递事件名称，第二个参数是事件发生时执行的函数。为简单起见，我们在这里全部写成alert形式。

现在准备工作都已就绪，在这些事件被触发时就可以看到效果了。可怎么触发事件呢？为了便于控制，我们设置3个按钮，在按钮按下时，就会触发相应的事件，如代码清单2-3所示。

代码清单2-3　触发person的事件

```
Ext.get('walk').on('click', function() {
    person.fireEvent('walk');
});

Ext.get('eat').on('click', function() {
    person.fireEvent('eat', '早餐', '中餐', '晚餐');
});

Ext.get('sleep').on('click', function() {
    person.fireEvent('sleep', new Date());
});
```

接着调用fireEvent()就会触发事件，传入一个事件名称作为参数，该事件对应的监听函数就会执行。

下面要讲的是只有动态语言才有的特性。如果想给监听方法传递参数，直接把参数写到fireEvent()里就行了，不用管参数数量，也不用管参数类型（字符串、数字、日期、数组等）。不过，需要确保监听函数可以处理你传递过去的参数。如果监听函数需要某个参数，触发事件时却忘了传递过去，这时就会报告错误。这种因为动态语言的特性而导致的非直观错误非常难以查找。

正如on是addListener的简写，removeListener的简写形式是un。你可以用它删除某个事件对应的监听函数，具体操作如下面的代码所示。

```
var fn = function() {
```

```
        // TODO:
    }
    person.on('walk', fn);
    person.un('walk', fn)
```

还有一个purgeListeners()函数，它可以把所有的监听器都删除掉。请注意，是所有的监听器，一旦调用了这个函数，所有的事件都不起作用了。因此，一定要慎用。

2.1.2 浏览器事件

浏览器事件即传统意义上的鼠标单击、移动等事件，是由浏览器根据用户的动作触发的，与页面元素紧密相关。EXT使用Ext.EventManager、Ext.EventObject和Ext.lib.Event对原生浏览器事件进行了封装，最后展现在我们面前的就是一套统一的跨浏览器的通用事件接口。

也许有的读者会问，HTML元素本身已经支持事件，为什么还要再重新设计一套事件机制？实际上，基本上所有的JavaScript框架都会实现自己的事件机制。原因很多，但是有一点很重要：HTML元素对事件的处理是通过简单的单一绑定实现的。也就是说，如果不进行任何封装，事件只能绑定到一个事件处理句柄，如下面的代码所示。

```
var e=document.getElementById("test");
e.onclick=function(){alert("handle1")};
e.onclick=function(){alert("handle2")};
```

运行上述代码后你会发现，单击test按钮后只会弹出一个显示handle2的提示框，因为第一个事件已经被第二个事件覆盖了。然而，使用EXT框架，你就不用担心这个问题，同一个事件可以依次被绑定到多个事件处理句柄上，如下面的代码所示。

```
Ext.onReady(function(){
    var test = Ext.get('test');
    test.on('click', function() {
        alert("handle1");
    });
    test.on('click', function() {
        alert("handle2");
    });
});
```

首先使用Ext.get('test')获得HTML中id="test"对应的按钮，然后使用on('click', function(){})的形式为它添加两个监听函数，这两个函数会在触发单击（click）事件时依次执行，不会出现覆盖的问题。

下面我们来单独分析一下EXT事件模型的几个主要组成部分，并讨论它们的常用功能。

2.1.3 Ext.lib.Event

Ext.lib.Event是定义在adapter中的工具类，它封装了不同浏览器的事件处理函数，为上层组件提供了统一的功能接口。

对于定义在adapter中的适配器工具，EXT自带的文档中没有任何关于这个类函数的说明，

而且在实际中也很少直接用到这个类，只是与事件相关的那些操作最后都会归结为对这些底层函数的调用。

Ext.lib.Event中定义了以下几个主要的函数。

- getX()、getY()、getXY()获得发生的事件在页面中的坐标位置，getXY()返回的是一个数组。如果希望获得对应的*x*、*y*坐标，则需要使用getXY()[0]和getXY()[1]的形式，getXY()[0]代表*x*坐标，getXY(0)[1]代表*y*坐标。
- getTarget()返回事件的目标元素，该函数用来统一IE和其他浏览器使用的ev.target和ev.srcElement。
- on()和un()我们之前已经提到过，on()用于将事件监听函数绑定到元素对应的事件上，un()则执行反操作，将事件监听函数从元素上清除，而purgeElement()函数会把元素上的所有事件都清除。
- preventDefault()函数用于取消浏览器对当前事件所执行的默认操作。例如，在自定义右键菜单时就需要使用这个函数，防止单击鼠标右键时弹出浏览器自身的右键菜单。
- stopPropagation()函数的作用是停止事件传递，这是与浏览器中HTML元素事件的传递机制相关的。内层的HTML元素触发的元素会传递外层的HTML元素，调用stopPropagation()函数会中断这个传递过程。
- 如果我们希望停止一个事件，可以调用stopEvent()函数。其内部调用了preventDefault()和stopPropagation()两个函数，取消了浏览器的默认操作，同时停止了事件传递。
- onAvailable()函数有3个参数：id、fn和scope。它的作用是等到id对应的HTML元素可用时才执行fn这个函数，scope表示调用函数的作用域。如果你仔细阅读代码就会了解，里边使用的setInterval()循环检测id对应的HTML元素，直到它可用时才停止循环执行fn函数。
- resolveTextNode()这个函数与事件没有关系，它仅仅是用来判断。如果参数node是一个文本节点，就返回它的上层节点；否则，返回node本身。这个函数是getTarget()函数中使用的工具函数。
- getRelatedTarget()函数会返回事件相关的HTML元素，它先尝试获得ev.relatedTarget。如果这个ev.relatedTarget属性不存在，就通过ev.type判断事件类型。如果ev.type == "mouseout"，就返回ev.toElement；如果ev.type == "mouseover"，就返回ev.fromElement。

至此，我们介绍了Ext.lib.Event中所有的主要函数，可以看到它们都提供了一些通用的底层调用方法，后面会在Ext.EventManager或Ext.EventObject中使用这些函数。

2.1.4　Ext.util.Observable

Ext.util.Observable在EXT的事件模型体系中有举足轻重的地位，位于EXT组件的顶端，为EXT组件提供处理事件的最基本功能。要实现一个可以处理事件的EXT组件，最直接的方法就

是继承Ext.util.Observable。

本章开始的示例已经展示了如何使用Ext.util.Observable实现自定义事件，你大概也了解了addListener/on、removeListener/un、addEvents、fireEvent这些函数的基本用法。这些函数的常用功能就不再赘述，下面主要讨论一下与事件有关的高级功能。

首先，当addListener/on用于注册事件时，可以使用复合式参数，如下面的代码所示。

```
Ext.get('test').on('click', fn, this, {
    single: true,
    delay: 100,
    testId: 4
});
```

上述代码中，在调用on()为id=test这个按钮注册click事件时使用了4个参数。click是事件名称，fn是在click事件被触发时执行的函数；this表示fn执行时的作用域是this，第四个参数就是我们所说的复合式参数了。

如果你已经运行过01-04a.html这个示例，就应该知道复合式参数产生了什么样的效果。示例的运行结果是，在第一次单击“test”按钮时，alert提示框不是立刻弹出，而是延迟一段时间才弹出。在第二次单击“test”按钮时，就什么也不会发生了。

这些效果就是复合式参数的作用，single:true表示这个注册的事件处理函数仅执行一次；delay:100表示函数会在事件发生后延迟100 ms才执行。

Ext.util.Observable支持的特殊参数还有一个buffer，如下面代码所示。

```
Ext.get('test').on('click', fn, this, {
    buffer: 1000,
    testId: 4
});
```

乍一看，buffer和delay的作用是相同的，都是延迟一段时间后执行对应的监听函数，但是buffer会创建一个Ext.util.DelayTask对象，并把fn放入其中等待执行。在等待的过程中，如果我们再次触发了事件，那么上次的任务就会被取消，并把新的fn放入任务队列里，这样就可以保证fn不会重复执行多次。

再来看看函数fn的内容，如下面的代码所示。

```
var fn = function(e, el, args) {
    alert("handle1");
    alert(args.testId);
};
```

这里演示了如何从事件监听函数中获得当初在复合式参数中定义的数据。我们在复合式参数中定义了一个testId:4，它会被fn函数的第三个参数传递到函数中，可以直接通过args.testId获得这个参数的值。

我们还可以在复合式参数中使用scope指定事件监听函数调用的作用域。不过因为在on()函数中已经有了scope参数，所以这个复合式参数的作用也就不那么重要了。

除了复合式参数，还可以使用on()一次定义多个事件监听器，如下面代码所示。

```
Ext.get('test').on({
    'click': {
        fn:fn
    },
    'mouseover': {
        fn: fn,
        single: true,
        delay: 100
    }
});
```

不仅如此，还可以使用复合式参数为监听函数设置更多功能。如上例中的single:true和delay:100，使得只有鼠标第一次移动到“test”按钮上时才会触发fn函数，并且在事件发生100 ms后才能执行fn函数。

而对于之前提及的复合式参数中的scope，在这种情况下就必不可少了。因为已经不再用原来的方式定义函数执行的scope作用域，所以只有使用复合式参数中的scope才能保证监听函数在正确的作用域上执行。

Ext.util.Observable还有一个重要的功能，就是可以为某个事件设置拦截器，统一管理方法的触发。我们使用capture()和releaseCapture()来实现这个功能。

我们在Person的示例中使用capture()函数拦截事件的触发，如下面的代码所示。

```
Ext.get('capture1').on('click', function() {
    Ext.util.Observable.capture(person, function() {
        alert('capture1');
        return true;
    });
});
```

单击“capture1”按钮时，拦截person的fireEvent()函数。在触发任意一个事件时，弹出alert提示框，并返回true。这样不会中止事件的发生，设置的监听函数还可以正常捕获到事件，并执行相应的处理操作，如下面的代码所示。

```
Ext.get('capture2').on('click', function() {
    Ext.util.Observable.capture(person, function() {
        alert('capture2');
        return false;
    });
});
```

单击“capture2”按钮时，弹出alert提示框后返回false，这会中止事件的发生，导致fireEvent()失效。监听函数无法捕获到事件，当然也就无法执行处理操作。这样就给我们一个选择的机会，通过控制capture()中处理函数的返回值来决定是继续执行某个事件的监听函数，还是直接中止该事件的发生。

我们可以为一个对象设置多个capture()拦截函数，这些拦截函数会形成一个处理链条，只要其中任何一个拦截函数返回false，就会中止处理过程。

releaseCapture()函数是capture()函数的反向操作，它会一次性清除fireEvent()上所

有的拦截函数，不过我们无法通过它准确删除某一个拦截函数。一旦执行了releaseCapture()，那么之前设置的所有拦截函数就都失效了。

如果只想通过一次设置来暂停某个对象中所有事件的发生，可以考虑使用suspendEvents()函数，如下面的代码所示。


```
Ext.get('suspendEvents').on('click', function() {
    person.suspendEvents();
});
```

调用suspendEvents()后，person中所有的事件都会失效，只有再次调用resumeEvents()才能取消这个效果，如下面的代码所示。

```
Ext.get('resumeEvents').on('click', function() {
    person.resumeEvents();
});
```

暂停（suspendEvents）与继续（resumeEvents）这两个事件函数可以帮助我们统一管理某一对象的事件。

2.1.5 Ext.EventManager

作为事件管理器，Ext.EventManager定义了一系列事件相关的处理函数，其中最常用的当属onDocumentReady、onWindowResize和onTextResize。其中onDocumentReady就是我们经常见到的Ext.onReady()，它会在页面文档渲染完毕但图片等还未下载时调用启动函数。

让我们来看一下Ext.onReady()的应用，如代码清单2-4所示。

代码清单2-4 使用Ext.onReady

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
        <title>grid</title>
        <link rel="stylesheet" type="text/css" href="../../resources/css/ext-all.css" />
        <script type="text/javascript" src="../../adapter/ext/ext-base.js"></script>
        <script type="text/javascript" src="../../ext-all.js"></script>
        <script type="text/javascript">
           Ext.onReady(function(){
               Ext.Msg.alert('信息', Ext.get('test'));
           });
        </script>
    </head>
    <body>
        <button id="test">test</button>
    </body>
</html>
```

打开01-05a.html后，可以在页面上看到图2-1的效果。

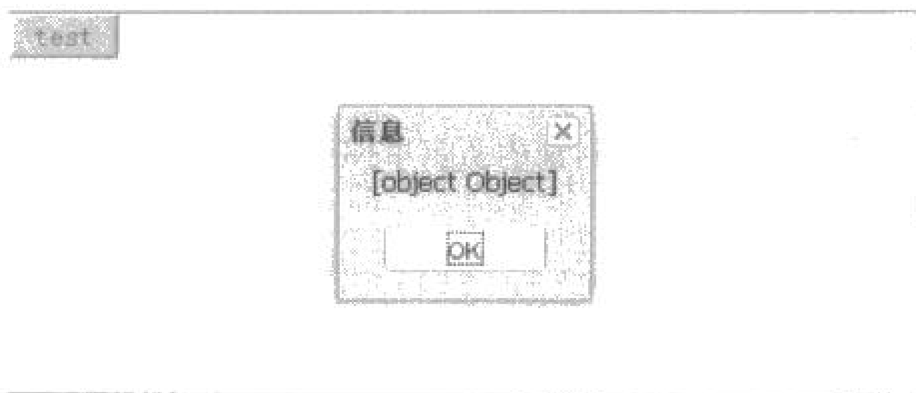


图2-1　使用Ext.onReady

Ext.get('test')正确获得了“test”按钮对应的Element对象，并显示出来了。从代码中可以看到，我们编写的脚本位于<head>标签中，而“test”按钮写在下部的<body>标签里。按照正常的加载执行顺序，脚本在<body>标签加载之前就执行了，这样Ext.get('test')就会找不到“test”按钮，也就无法显示正常的信息。但是，在示例中我们得到了正常的结果，说明Ext.onReady()可以保证它里面的内容会在所有的HTML元素都加载完成后才执行。这样就避免了许多加载顺序导致的问题，我们也不用再为脚本需要放在对应的HTML元素之后而费神了。

虽然我们说没有使用Ext.onReady()可能会导致种种问题，但还是耳听为虚。现在让我们看一下，假如没有Ext.onReady()的帮助会出现什么问题，如图2-2所示。

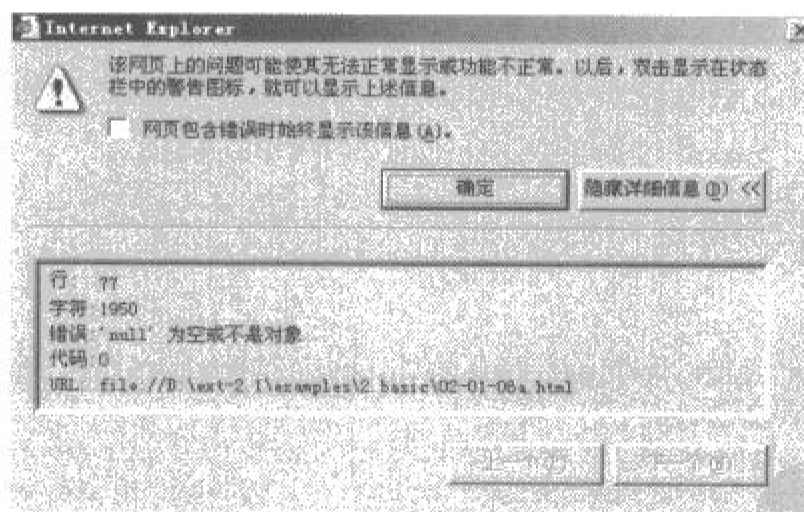


图2-2　不使用Ext.onReady

页面没有显示任何效果，而且出现了一个错误提示。这是因为脚本执行时页面标签还没有加载完全，EXT无法获得对应的HTML元素对自身进行渲染，需要的元素为null，结果就会报告这个错误。

所以，无论什么时候，都要使用Ext.onReady()来确保脚本是在HTML元素都加载完成后才执行，这样可以避免很多潜在的问题。

onWindowResize()的作用是监听浏览器窗口的大小改变，它会提醒我们浏览器窗口的大小在何时发生了改变，以及改变后的大小，如下面的代码所示。

```
Ext.EventManager.onWindowResize(function(width, height) {
```

```
        alert('width:' + width + ', height:' + height);
    });
```

不必将它放在Ext.onReady()中，因为它监听的是window对象，这个对象在页面一打开时就存在了。onWindowResize()的监听函数只有两个参数width和height，分别代表当前窗口的宽和高。

onTextResize()是一个有用的工具函数，它可以监听用户修改浏览器的文字大小，如下面的代码所示。

```
Ext.onReady(function() {
    Ext.EventManager.onTextResize(function(oldSize, newSize) {
        alert('oldSize:' + oldSize + ', newSize:' + newSize);
    });
});
```

onTextResize()会在<body>标签中添加一个包含测试文字的div标签，然后监听测试文字的大小。如果测试文字的大小发生了改变，就执行监听函数，并发送给监听函数两个参数，第一个参数oldSize是改变前文字的大小，第二个参数newSize是改变后文字的大小。

需要注意的是，因为onTextResize()需要操作<body>标签向其中添加测试用的元素，所以它必须包含在Ext.onReady()中，否则会出现错误。

2.1.6 Ext.EventObject

Ext.EventObject是对事件的封装，它将EXT自定义事件和浏览器事件结合在一起使用。除此之外，它还提供了丰富的辅助工具函数，帮助我们获得事件相关的信息。

Ext.EventObject定义了一系列的功能按键，处理按键事件时不用再去硬背ASCII码了，这些功能按键如表2-1所示。

表2-1 Ext.EventObject中按键的映射关系

Ext.EventObject中的名称	ASCII码	Ext.EventObject中的名称	ASCII码
BACKSPACE	8	PAGEDOWN	34
TAB	9	END	35
RETURN	13	HOME	36
ENTER	13	LEFT	37
SHIFT	16	UP	38
CONTROL	17	RIGHT	39
ESC	27	DOWN	40
SPACE	32	DELETE	46
PAGEUP	33	F5	116

下面利用一个input type=text的输入文本框来演示监听用户按键事件，并在用户输入空格时提示，如下面的代码所示。

```
Ext.get('text').on('keypress', function(e) {
    if (e.charCode == Ext.EventObject.SPACE) {
```

```
            Ext.Msg.alert('info', '空格');
        }
    });
```

这个示例在用户按下空格时弹出alert提示框。注意监听函数的参数e，其实它就是一个Ext.EventObject的实例，它的charCode代表了刚才按下按钮的ASCII码。我们将它与Ext.EventObject.SPACE相比较，判断当前按键是否是空格。

Ext.EventObject是对事件的封装，如果我们想得到原始的浏览器事件，可以通过Ext.EventObject的browserEvent来获得。不过，这可是未经任何加工处理的原生事件，在不同的浏览器上可能存在着很大的差异，所以还是建议使用Ext.EventObject。

通过Ext.EventObject的文档可以了解到，它包含的许多工具函数都与Ext.lib.Event中的函数是重名的，如getPageX()、getPageY()、getPageXY()、getTarget()和getRelated-Target()等。这些函数实际上都是通过Ext.lib.Event实现的，Ext.EventObject代表浏览器事件，在内部使用Ext.lib.Event处理对应的browserEvent。

Ext.EventObject对Ext.lib.Event扩展的部分是对鼠标事件和按键事件的增强，它可以判断ALT、CTRL、SHIFT这些功能键是否被按下。既可以单独使用altKey、ctrlKey和shiftKey判断是否有功能键被按下，也可以使用hasModifier()判断是否有功能键被按下。这个功能一般要与其他的按键状态相配合，用于判断组合按键的情况。

Ext.EventObject提供的另一个有趣的功能函数名称为getWheelDelta()，可以获得鼠标滚轮的delta值。在下面的示例中，我们监听了mousewheel事件，在滚轮转动时动态修改test这个div的宽度，这样又给我们提供了一种提升用户体验的方法，如下面的代码所示。

```
Ext.get(document.body).on('mousewheel', function(e) {
    var delta = e.getWheelDelta();
    var test = Ext.get('test');
    var width = test.getWidth();
    test.setWidth(width + delta * 500, true);
});
```

2.2　EXT的核心组件

接下来介绍的是EXT中的核心组件，这些核心组件作为EXT整体架构的基石，提供了基本的生命周期、布局方式管理。

2.2.1　Ext.Component

Ext.Component是EXT中所有组件的基类，它的所有子类都自动享有了标准EXT组件的生命周期，包括创建、渲染和销毁。它们也自动支持了标准的隐藏/显示以及启用/禁用等操作。

图2-3是一棵单根的组件继承树，从Ext.Component开始依次延展开来，每个组件都对应一个xtype属性，在EXT中可以通过xtype属性直接指定在某处使用的组件。

所有组件都允许在Ext.Container及其子类中进行延迟渲染（lazy render），也可以把组件注

册到Ext.ComponentMgr中，这样就可以在任何地方使用Ext.getCmp()函数直接根据id获得对应的组件。

所有可视化的组件都是Ext.Component的子类，这样我们就能把它们放到layout中进行布局管理。



图2-3 组件继承树

组件大致可以分成3类：基本组件（见表2-2）、工具条组件（见表2-3）和表单组件（见表2-4）。

表2-2 基本组件

xtype	组件名称	描 述
box	Ext.BoxComponent	具有边框属性的组件
button	Ext.Button	按钮
colorpalette	Ext.ColorPalette	调色板
component	Ext.Component	组件
container	Ext.Container	容器
cycle	Ext.CycleButton	循环按钮
dataview	Ext.DataView	数据显示视图
datepicker	Ext.DatePicker	日期选择面板
editor	Ext.Editor	编辑器
editorgrid	Ext.grid.EditorGridPanel	可编辑的表格
grid	Ext.grid.GridPanel	表格
paging	Ext.PagingToolbar	分页工具条
panel	Ext.Panel	面板（可进行子布局）
Progress	Ext.ProgressBar	进度条
splitbutton	Ext.SplitButton	可下拉的按钮
tabpanel	Ext.TabPanel	选项面板
treepanel	Ext.tree.TreePanel	树
viewport	Ext.ViewPort	视图
window	Ext.Window	窗口

表2-3 工具条组件

xtype	组件名称	描 述
toolbar	Ext.Toolbar	工具条
tbfill	Ext.Toolbar.Fill	右对齐填充'->'
tbitem	Ext.Toolbar.Item	工具条项目
tbseparator	Ext.Toolbar.Separator	工具条分隔符'-'
tbspacer	Ext.Toolbar.Spacer	工具条空白
tbtext	Ext.Toolbar.TextItem	工具条文本项

表2-4 表单组件

xtype	组件名称	描 述
form	Ext.FormPanel	表单面板
checkbox	Ext.form.Checkbox	多选框
combo	Ext.form.ComboBox	下拉列表
datefield	Ext.form.DateField	日期选择项
field	Ext.form.Field	输入框
fieldset	Ext.form.FieldSet	组
hidden	Ext.form.Hidden	表单隐藏域

（续）

xtype	组件名称	描　述
htmleditor	Ext.form.HtmlEditor	HTML编辑器
numberfield	Ext.form.NumberField	数字编辑器
radio	Ext.form.Radio	单选框
textarea	Ext.form.TextArea	区域文本框
textfield	Ext.form.TextField	表单文本框
timefield	Ext.form.TimeField	时间录入项
trigger	Ext.form.TriggerField	触发录入项

通常会在进行布局时借助xtype实现简化配置和延迟加载的功能，如果希望了解如何在布局中使用xtype，可以阅读第8章的相关内容。

2.2.2 Ext.BoxComponent

Ext.BoxComponent也是一个比较重要的基础类，它直接继承自Ext.Component，并实现了定位和控制自身大小的功能。

可以使用pageX、pageY、x、y为Ext.BoxComponent指定具体的坐标，也使用width和height为Ext.BoxComponent指定长度和宽度，或者使用autoHeight和autoWidth让Ext.BoxComponent根据本身的内容自动调整长度和高度。

下面演示如何使用Ext.BoxComponent在页面中定义位置和大小，如下面的代码所示。

```
var box = new Ext.BoxComponent({
    el: 'test',
    style: 'background-color:red;position:absolute;',
    pageX: 100,
    pageY: 50,
    width: 200,
    height: 150
});
box.render();
```

图2-4演示了上述代码的显示效果。

所以，如果需要制作一个可控制大小和位置的组件，可以直接从Ext.BoxComponent继承。

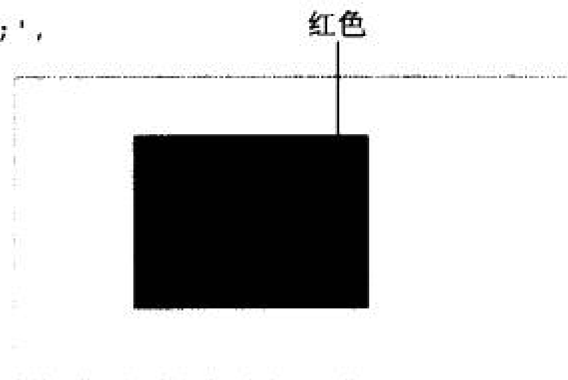


图2-4　Ext.BoxComponent

2.2.3 Ext.Container

Ext.Container继承自Ext.BoxComponent，它提供了两个重要的参数layout和items。layout参数指定当前组件使用何种布局，items参数中包含的是当前组件中的所有子组件。

如果你想制作一个可以对自身包含的子组件进行布局的组件，那么就需要继承Ext.Container，Ext.Container是一切可布局组件的超类，图2-5是它的继承树形图。

有关Ext.Container的更多细节，请参考第8章中有关布局的部分。

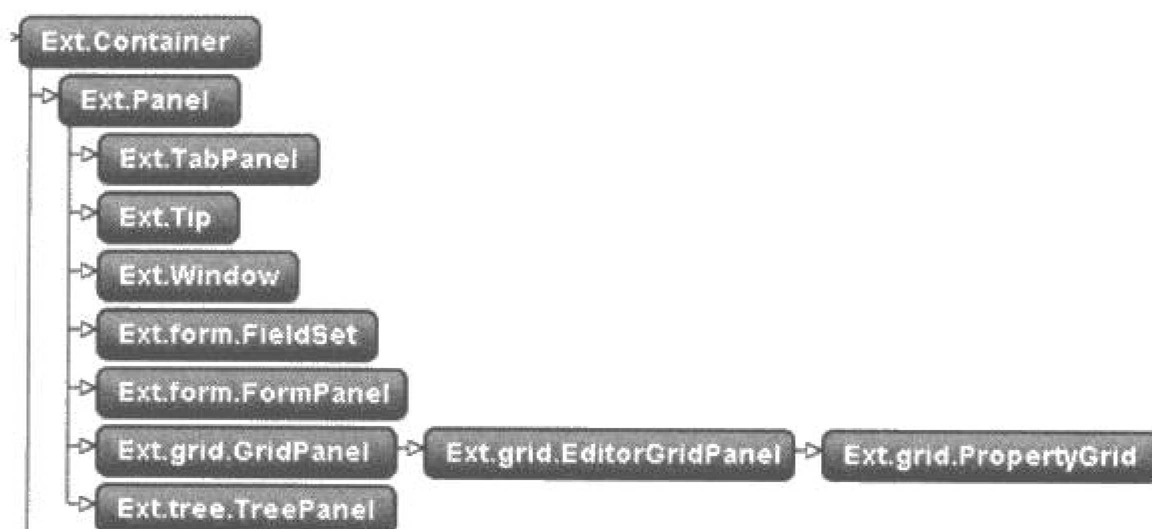


图2-5　Ext.Container的继承树形图

2.2.4 Ext.Panel

Ext.Panel是EXT中经常用到的一个组件，它直接继承自Ext.Container。与上面那些组件不同的是，Ext.Panel无需继承就可以直接使用。我们可以使用title参数定义它显示的标题，使用tbar和bbar设置上下位置的工具条，使用collapseFirst、collapsed、collapsedCls和collapsible设置与面板折叠相关的配置。除此之外，还可以使用floating和shadow设置浮动阴影效果，以及使用HTML直接设置面板内容。

现在我们来设置一个包含浮动阴影的、可拖放的、可折叠的、设置了大小、位置、标题和内容的Ext.Panel，如下面的代码所示。

```
var panel = new Ext.Panel({
    el: 'test',
    title: '测试标题',
    floating: true,
    shadow: true,
    draggable: true,
    collapsible: true,
    html: '测试内容',
    pageX: 100,
    pageY: 50,
    width: 200,
    height: 150
});
panel.render();
```

其显示效果如图2-6所示。

可见，Ext.Panel是一个相当完美的标准组件。



图2-6　Ext.Panel

2.2.5 Ext.TabPanel

Ext.TabPanel是另一个常用组件，一两句话很难说清楚，但是看到显示效果就很容易理解

了。我们先来看图2-7。

如图2-7所示，显示的内容随着你点击的标题而变化。实际上，它是多个不同内容的容器，任意组件直接使用add()函数便可添加到Ext.TabPanel中。如果不特别指定xtype，就会默认使用Ext.Panel为这些内容生成子面板，如代码清单2-5所示。



图2-7 Ext.TabPanel

代码清单2-5 创建Ext.TabPanel

```
var tabs = new Ext.TabPanel({
    renderTo: document.body,
    height: 100
});

tabs.add({
    title: '标题1',
    html: '内容1'
});
tabs.add({
    id: Ext.id(),
    title: '标题2',
    html: '内容2',
    closable: true
});

tabs.activate(0);
```

首先创建一个Ext.TabPanel，第一个参数指定它渲染的元素，我们直接把它放到HTML页面的Body部分。

然后使用add函数向TabPanel里添加两个标签，同时添加对应的内容。我们来看一下它里边的参数。

- 第一个参数是这个标签对应的id，我们使用Ext.id()函数生成唯一的id值。
- 第二个参数是标签上显示的标题。
- 第三个参数是单击标签后显示的内容。实际上这里可以放置任何HTML，你可以把所有想放的内容放到里边。
- 最后的参数是一个布尔值：true或false，这个参数决定了生成的标签是否可以手工关闭，默认是false，不会显示关闭按钮。当手工设置为true后，标签上显示一个关闭按钮，单击关闭按钮后，标签和对应的内容被关掉。

添加标签后，调用activate()函数让指定的标签变成激活状态，参数是一个从0开始的索引值，activate(0)激活第一个标签。

现在，这些参数统一放到大括号里，这样我们更容易理解这些参数的含义，比如renderTo是渲染的元素，height是生成的高度，title是对应标题，HTML是对应内容，用closable确定这些标签是否能关闭。tab.activate(0)依然是激活第一个标签。

上述代码对应的示例在02-05a.html中。

接下来，我们可以向添加的标签里多放一些内容，创建两个按钮，一个按钮新建包含表格的标签，另一个按钮新建包含Panel的标签，如图2-8所示。

其实只要知道一点就可以，EXT 3.0的布局里可以随意控制这些。比如使用items:[grid]指定在标签里放生成好的表格，但记得要写上layout:'fit'。这个布局形式可以让内部的表格随着外部而自然展开，否则得到的只是一个高度为零的表格，什么也看不到。

加入表格的代码如下所示：

```
var tab =  tabs.add({
    title: '表格' + id,
    closable: true,
    layout: 'fit',
    items: [grid]
});
tabs.activate(tab);
```

上面的代码在添加标签时自动生成一个标题（title）属性，设置closable:true，则可以手工关闭这个标签。layout:'fit'让里边自动填充整个空间，最后items:[grid]告诉我们这里面放的是表格，当然这个表格是我们之前做好的。最后调用tabs.activate(tab)激活刚才添加的标签，这样就可以看到结果了。

该示例在02-05b.html中。

上面的例子中我们都是通过JavaScript为TabPanel创建显示内容，如果我们需要的内容是从后台得到的HTML该怎么做呢？如果这个HTML里有自己的JavaScript脚本，是不是可以在加载完成后再执行这些脚本呢？

实际上可以通过设置autoLoad参数来实现这个功能，这些标签默认是延迟加载（lazy load）的，只有在激活时才使用Ajax到后台获取数据，如图2-9所示。

图2-8 动态Ext.TabPanel

图2-9 通过设置autoLoad来从后台获取数据

这里会显示乱码是因为Ajax中应该使用UTF-8编码，而中文页面用GBK编码，所以就会有问题。如果想正常显示中文，需要把那些文件改成UTF-8编码，就不会再出现乱码了。

这部分代码很简单，把HTML去掉，换成autoLoad即可，如下面的代码所示。

```
tabs.add({
    title: '标题1',
    autoLoad: {url: '02-05c1.html'}
});
```

不过，如果这样做，在02-05c1.html里有JavaScript脚本时，代码也不会执行。如果想执行包含在HTML里的脚本，还需要加一个scripts属性，如下面的代码所示。

```
tabs.add({
    title: '标题2',
    autoLoad: {url: '02-05c2.html', scripts: true},
    closable: true
});
```

现在，02-05c2.html里的脚本就可以执行了。02-05c.html里有两个按钮，第一个按钮添加没有脚本的02-05c1.html，第二个按钮添加有脚本的02-05c2.html。

上面主要讲解了EXT中的基本事件模型以及一些常用组件的功能和方法。在下面几章中，我们会分别介绍各个组件的使用方法和EXT为我们提供的更丰富的功能。

2.3　小结

本章主要介绍了EXT的事件模型和核心组件，并通过实例详细讲解了事件模型的使用方法。此外，还展示了EXT中的组件继承图，并对其中的常用组件进行了详细讲解。

第3章 表格控件

3

本章内容

- 表格的特性简介
- 制作一个简单的表格
- 表格常用功能详解
- 表格渲染
- 给表格的行和列设置颜色
- 自动显示行号和复选框
- 选择模型
- 表格视图——Ext.grid.GridView
- 表格分页
- 后台排序
- 可编辑表格控件——EditorGrid
- 属性表格控件——PropertyGrid
- 分组表格控件——Group
- 可拖放的表格
- 表格与右键菜单

3.1 表格的特性简介

EXT中的表格功能非常强大，包括排序、缓存、拖动、隐藏某一列、自动显示行号、列汇总、单元格编辑等实用功能。

表格由类Ext.grid.GridPanel定义，继承自Ext.Panel，其xtype为grid。在EXT中，表格控件必须包含列定义信息，并指定表格的数据存储器。表格的列信息由类Ext.grid.ColumnModel定义，而表格的数据存储器由Ext.data.Store定义。根据解析的数据不同，数据存储器可分为JsonStore、SimpleStroe、GroupingStore等。

接下来要讨论表格的各种功能，包括选择一条记录、选择多条记录、突出显示选中的行、调整列宽度、列排序、显示行号、支持复选框、设置查看某些列，以及支持本地和远程分页。还有可编辑的表格、添加新行、删除一或多行、数据校验、拖放改变表格大小、在表格里拖放一或多

行，甚至还可以在树形和表格之间进行数据拖放，这些功能竟然都在EXT表格控件里实现了，令人惊叹！

3.2 制作一个简单的表格

不管学习何种技术，只了解概念是不行的，一定要实践。下面我们在examples中的示例的基础上自己制作一个表格，从中可以知道表格需要进行哪些配置。

首先，表格是二维的。与在数据库中新建表一样，我们要先设置表的列数、每列的名称和类型，以及如何显示。表格的结构和数据库表的结构非常类似。

在EXT中，列的定义叫做ColumnModel，简称为cm，它是整个表格的列模型，应该首先创建。

在这里，我们创建一个包含3列的表格（如下面的代码所示），第1列是编号（id），第2列是名称（name），第3列是描述（descn）。

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},`
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
]);
```

var cm = new Ext.grid.ColumnModel(...)负责创建表格的列信息。表格包含的列由columns配置属性来描述，简称cm。columns是一个数组，每一行数据元素描述表格的一列信息，表格的列信息包含首部显示文本（header）、列对应的记录集字段（dataIndex）、列是否可排序（sortable）、列的渲染函数（renderer）、宽度（width）、格式化信息（format）等，在上面的示例中只用到了header及dataIndex。

表格的结构确定后，我们就可以向里面添加数据了。当然，数据也是二维的，为简便起见，我们参照examples里的array-grid.js中的方式，把数据直接写到JavaScript里，如下面的代码所示。

```
var data = [
    ['1','name1','descn1'],
    ['2','name2','descn2'],
    ['3','name3','descn3'],
    ['4','name4','descn4'],
    ['5','name5','descn5']
];
```

在上面的代码中，var data=...用来定义表格中要显示的数据，这是一个有5条记录的二维数组，显示到表格里就应该是5行，每行3列，正好对应id、name和descn。此时，我们应该可以想像出表格显示的结果了。为了让美好的愿望变成现实，我们还需要转化原始数据，如下面的代码所示。

```
var store = new Ext.data.Store({
   proxy: new Ext.data.MemoryProxy(data),
   reader: new Ext.data.ArrayReader({}, [
        {name: 'id'},
        {name: 'name'},
        {name: 'descn'}
    ])
});
```

```
store.load();
```

var store=...用来创建一个数据存储对象，这也是表格必须配置的属性，数据存储对象store负责把各种各样的原始数据(如二维数组、JSON对象数组、XML文本等)转换成dExt.data.Record类型的对象。通过Ext.data.Store，我们可以把任何格式的数据转化成表格可以使用的形式，这样就不需要为每种数据格式写一个对应的实现。

store对应两个部分：proxy和reader。proxy是指获取数据的方式，reader是指如何解析这一堆数据。这里我们用的是Ext.data.MemoryProxy，它是专门用来解析JavaScript变量的。在定义MemoryProxy对象时，只需要把上面定义的data作为参数传递进去即可。

Ext.data.ArrayReader专门用来解析数组，并且告诉我们它会按照定义的规范进行解析，定义3个名称：id、name和descn。再看前面cm定义的地方，这里的3个名称就是和cm里的dataIndex相对应的。这样，cm就知道column是如何与store中的数据相对应的。注意，要执行一次store.load()，来初始化数据。

如果第1列数据不是id而是name，第2列数据不是name而是id，那么就要使用mapping来指定，如下面的代码所示。

```
var store = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id', mapping: 1},
        {name: 'name', mapping: 0},
        {name: 'descn', mapping: 2}
    ])
});
```

结果如图3-1所示。

在图3-1中，id和name两列的数据显示调换了。所以，无论数据排列顺序如何，我们都可以使用mapping来控制对应关系。唯一需要注意的是，索引是从0开始的，所以对应第1列要写成mapping:0，依次类推。

名称	编号	描述
name1	1	descn1
name2	2	descn2
name3	3	descn3
name4	4	descn4
name5	5	descn5

图3-1　使用mapping指定数据对应位置

表格的列模型定义好了，原始数据和数据的转换也已经完成，剩下的只需要把它们装配在一起，我们的表格就创建成功了，如下面的代码所示。

```
var grid = new Ext.grid.GridPanel({
    renderTo: 'grid',
    store: store,
    cm: cm
});
```

Ext.grid.Grid的renderTo属性指示EXT将表格渲染到什么地方，所以，在HTML里应该有一个<div id="grid"></div>与之对应。

把所有代码组合到一起（见代码清单3-1），看看效果吧。

代码清单3-1 创建Ext.grid.GridPanel的完整代码

```
    var cm = new Ext.grid.ColumnModel([
        {header:'编号',dataIndex:'id'},
        {header:'名称',dataIndex:'name'},
        {header:'描述',dataIndex:'descn'}
    ]);

    var data = [
        ['1','name1','descn1'],
        ['2','name2','descn2'],
        ['3','name3','descn3'],
        ['4','name4','descn4'],
        ['5','name5','descn5']
];

    var store = new Ext.data.Store({
        proxy: new Ext.data.MemoryProxy(data),
        reader: new Ext.data.ArrayReader({}, [
            {name: 'id'},
            {name: 'name'},
            {name: 'descn'}
        ])
    });
    store.load();

    var grid = new Ext.grid.GridPanel({
        renderTo: 'grid',
        store: store,
        cm: cm
    });
```

下面就是一个显示数据的简单的表格，如图3-2所示。

该示例在03.grid/03-01.html文件中。

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

图3-2 一个简单的表格

3.3 表格常用功能详解

本节介绍在使用表格过程中经常会使用到的常用功能，使用这些功能我们可以迅速创建出一个强大而又漂亮的表格。

3.3.1　部分属性功能

默认情况下，表格可以拖放列（见图3-3），也可以改变列的宽度。如果要禁用这两个功能，在定义表格对象时将enableColumnMove和enableColumnResize设置为false即可。

默认情况下，表格也支持按住Shift和Ctrl键选择多行的功能（见图3-4）。全选后，只需要任意单击其中的一行，就可取消全选，只选中刚才单击的那一行。

编号	名称	描述
1	name1	descn1
2	name2	d 编号
3	name3	descn3
4	name4	descn4
5	name5	descn5

图3-3　在表格中拖放列

名称	描述	编号
name1	descn1	1
name2	descn2	2
name3	descn3	3
name4	descn4	4
name5	descn5	5

图3-4　一次选择多行

如果想让表格显示图3-5所示的斑马线效果，可以加上stripeRows:true，如下面的代码所示。

```
var grid = new Ext.grid.GridPanel({
//enableColumnMove: false,
//enableColumnResize: false,
    renderTo: 'grid',
    stripeRows: true,
    store: store,
    cm: cm
});
```

该示例在03.grid/03-01-03.html中。

表格还支持一种读取数据时的遮罩和提示功能（见图3-6），设置属性loadMask: true，在store.load()完成之前会一直显示“Loading...”。

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

图3-5　斑马线效果

编号　名称　描述

Loading...

图3-6　表格读取数据时的提示功能

图3-6的具体实现如下面的代码所示。

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
```

```
]);

var data = [
    ['1','name1','descn1'],
    ['2','name2','descn2'],
    ['3','name3','descn3'],
    ['4','name4','descn4'],
    ['5','name5','descn5']
];

var store = new Ext.data.Store({
    proxy: new Ext.data.ScriptTagProxy({
             url:'http://www.family168.com/data.json'}),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id'},
        {name: 'name'},
        {name: 'descn'}
    ])
});

var grid = new Ext.grid.GridPanel({
    renderTo: 'grid',
    width: 350,
    height: 150,
    loadMask: true,
    store: store,
    cm: cm
});
store.load();
```

在03.grid/03-01-04.html中，为了演示这一效果，没有使用MemoryProxy，因为用本地数据读取太快，根本看不到读取过程。所以我们使用了另一个远程读取数据的对象ScriptTagProxy，并将URL属性设置为http://www.family168.com/，这样返回的数据肯定是不符合要求的，于是读取提示信息将会始终停留在页面上，大家使用时也要注意这个问题。

3.3.2 自主决定每列的宽度

到这里，估计大家都会想，如果所有列的宽度都一样，那将会多么不方便，因为当列不够宽时，还要自己动手调整列的宽度。其实，cm支持给每列设置宽度，如果不设置，它会取一个默认值，默认宽度是100px。

要自定义宽度，只需设置该列的width属性即可，如下面的代码所示。

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id',width:20},
    {header:'名称',dataIndex:'name',width:80},
    {header:'描述',dataIndex:'descn',width:200}
]);
```

原来等宽的表格就会变成如图3-7所示的样子。

该示例在03.grid/03-02-01.html中。

当然，这样还是会比较麻烦，需要自己去计算每列的宽度。如果想让每列自动填满表格，只需要viewConfig中的forceFit即可。顾名思义，它是给GridView用的配置，它指示视图层重新计算所有列宽后填充表格，如图3-8所示。

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

图3-7　自定义每列的宽度

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

图3-8　forceFit自动延伸

实现代码如下所示。

```
var grid = new Ext.grid.GridPanel({
    renderTo: 'grid',
    store: store,
    cm: cm,
    viewConfig: {
        forceFit: true
    }
});
```

使用了forceFit以后，表格会根据cm里设置的width按比例分配，非常智能。当改变某一列的宽度时，表格将会重新计算其他列的宽度，既不会超出，也不会出现过多空余。

大家可能会发现，即使设置了forceFit，表格的右边仍然会留出一小段空白区域。实际上，这里的空白区域是为纵向滚动条保留的，右边的空白区域正好比纵向滚动条宽一点儿，在高度超出屏幕或表格高度时可以保证只出现纵向滚动条，而不会出现横向滚动条。

该示例在03.grid/03-02-02.html中。

除了使用autoSizeColumns/forceFit重新计算全部宽度以外，还可以考虑autoExpand-Column，它可以让指定列的宽度自动伸展，从而填充整个表格，如下面的代码所示。

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id',width:20},
    {header:'名称',dataIndex:'name',width:80},
    {id:'descn',header:'描述',dataIndex:'descn',width:200}
]);

var grid = new Ext.grid.Grid({
    renderTo: 'grid',
    store: store,
    cm: cm,
    autoExpandColumn: 'descn'
});
```

autoExpandColumn只能指定一列的id。注意，必须是id，原来我们设置的cm里都没有id，

现在为了使用autoExpandColumn，要给cm的descn设置id。于是在渲染时descn就可以自动延伸，直至充满整个表格，如图3-9所示。

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

图3-9 autoExpandColumn自动延伸

该示例在03.grid/03-02-03.html中。

3.3.3 让表格支持按列排序

在EXT中可以很方便地实现排序功能，只需要在定义列模型时增加sortable属性，如下面的代码所示。

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id',sortable:true},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
]);
```

设置sortable属性为true就表示该列允许排序，改动后的效果如图3-10所示。

示例在03.grid/03-03-01.html中。

你会发现编号的标题上多了一个向下的小箭头，表格里的数据按照编号降序排列。一切如此简单，我们已经实现了按列排序，如图3-11所示。

编号 ▾	名称	描述
5	name5	descn5
4	name4	descn4
3	name3	descn3
2	name2	descn2
1	name1	descn1

图3-10 按列排序

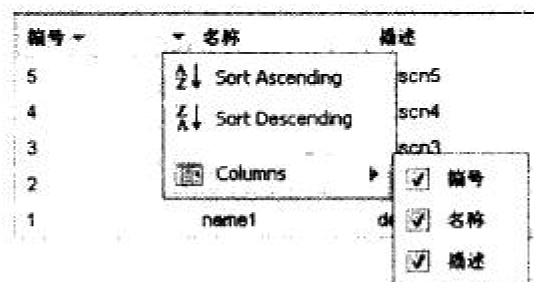


图3-11 列功能菜单

只要给每列加上sortable属性并且设置为true，便可以实现排序功能。同样，取消排序功能只要去掉sortable属性，或者将它设置为false即可，如下面的代码所示。

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id',sortable:true},
    {header:'名称',dataIndex:'name',sortable:true},
    {header:'描述',dataIndex:'descn',sortable:true}
]);
```

3.3.4 解决中文排序

国际上都使用ASCII码进行排序，而我们却按拼音顺序排序，EXT自动排好的中文在我们看起来却是一团糟。

例如我们在小学时常念的“啊啵呲嘚咯嗑嘿佛”，如下面的代码所示。

```
var data = [
    ['1','啊','descn1'],
    ['2','啵','descn2'],
    ['3','呲','descn3'],
    ['4','嘚','descn4'],
    ['5','咯','descn5']
];
```

为了可以立刻看到排序效果，我们通过sortInfo属性来为Ext.data.Store设置一个默认的排序方式，如下面的代码所示。

```
var store = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id'},
        {name: 'name'},
        {name: 'descn'}
    ]),
    sortInfo: {field: "name", direction: "ASC"}
});
```

注意多出来的那行（倒数第2行），它对应的是一个JsonObject，field代表排序的列名，direction代表排序方式，ASC是升序，DESC是降序。

结果你会看到，默认的排序方式会把数据显示变成图3-12这样，完全乱套了。

编号	名称 ▲
3	呲
5	咯
1	啊
2	啵
4	嘚

图3-12 默认排序效果

为了让表格实现中文排序的功能，我们需要重写Ext.data.Store的applySort函数，代码如下所示。

```
Ext.data.Store.prototype.applySort = function() {
    if (this.sortInfo && !this.remoteSort) {
        var s = this.sortInfo, f = s.field;
        var st = this.fields.get(f).sortType;
        var fn = function(r1, r2) {
            var v1 = st(r1.data[f]), v2 = st(r2.data[f]);
            if (typeof(v1) == "string") {
                return v1.localeCompare(v2);
            }
            return v1 > v2 ? 1 : (v1 < v2 ? -1 : 0);
        };
        this.data.sort(s.direction, fn);
        if(this.snapshot && this.snapshot != this.data) {
            this.snapshot.sort(s.direction, fn);
        }
    }
};
```

以上这段代码重写了Ext.data.Store的applySort函数，使其支持中文排序。你可以把这段代码加到ext-all.js文件的最后，或者放到HTML页面的最上面，总之是要在EXT初始化之后，实际代码调用之前执行。

现在我们可以看到排序结果和预想的是一样的了，如图3-13所示。

编号	名称 ▲
1	啊
2	啵
3	呲
4	嘚
5	咯

图3-13 中文排序

完整代码如下所示：

```
Ext.data.Store.prototype.applySort = function() {
    if (this.sortInfo && !this.remoteSort) {
        var s = this.sortInfo, f = s.field;
        var st = this.fields.get(f).sortType;
        var fn = function(r1, r2) {
            var v1 = st(r1.data[f]), v2 = st(r2.data[f]);
            if (typeof(v1) == "string") {
                return v1.localeCompare(v2);
            }
            return v1 > v2 ? 1 : (v1 < v2 ? -1 : 0);
        };
        this.data.sort(s.direction, fn);
        if(this.snapshot && this.snapshot != this.data) {
            this.snapshot.sort(s.direction, fn);
        }
    }
};

Ext.onReady(function(){

    var cm = new Ext.grid.ColumnModel([
        {header:'编号',dataIndex:'id',sortable:true,width:35},
        {header:'名称',dataIndex:'name',sortable:true,width:80},
        {id:'descn',header:'描述',dataIndex:'descn',sortable:true,width:200}
    ]);

    var data = [
        ['1','啊','descn1'],
        ['2','啵','descn2'],
        ['3','呲','descn3'],
        ['4','嘚','descn4'],
        ['5','咯','descn5']
    ];

    var store = new Ext.data.Store({
        proxy: new Ext.data.MemoryProxy(data),
        reader: new Ext.data.ArrayReader({}, [
            {name: 'id'},
            {name: 'name'},
            {name: 'descn'}
        ]),
        sortInfo: {field: "name", direction: "ASC"}
    });
    store.load();

    var grid = new Ext.grid.GridPanel({
```

```
        autoHeight: true,
        renderTo: 'grid',
        store: store,
        cm: cm,
        autoExpandColumn: 'descn'
    });

});
```

该示例在03.grid/03-04-01.html中。

3.3.5 显示日期类型数据

尽管返回的JSON里都是数字和字符串，但在EXT里我们同样可以从后台取得日期类型的数据，然后交给表格进行格式化。下面我们来看看如何实现它。

首先，定义一组数据，其中最后一列是日期格式的数据，如下面的代码所示。

```
var data = [
    ['1','name1','descn1','1970-01-15T02:58:04'],
    ['2','name2','descn2','1970-01-15T02:58:04'],
    ['3','name3','descn3','1970-01-15T02:58:04'],
    ['4','name4','descn4','1970-01-15T02:58:04'],
    ['5','name5','descn5','1970-01-15T02:58:04']
];
```

接着在reader里增加一行配置，除了设置name以外，还设置了type和dateFormat两个属性。其中type属性会告诉reader在解析原始数据时把对应的列作为日期类型处理，dateFormat属性把得到的字符串转换成相应的日期格式。按照EXT的约定，Y是年，m是月，d是日期，H是小时，i是分钟，s是秒，如下面的代码所示。

```
var store = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id'},
        {name: 'name'},
        {name: 'descn'},
        {name: 'date', type:'date',dateFormat:'Y-m-dTH:i:s'}
    ])
});
```

同样地，我们还需要在cm里增加一行配置，并且配置了renderer属性用于格式化日期格式的数据，代码如下所示。

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'},
    {header:'日期',dataIndex:'date',type:'date', renderer: Ext.util.Format.
    dateRenderer('Y年m月d日')}
]);
```

我们注意到，renderer属性对应的值是Ext.util.Format.dateRenderer('Y年m月d日')，

这就是EXT提供的日期格式化方法。这样我们就不需要再专门编写格式化日期的函数了，直接使用封装好的工具类即可。

该示例在03.grid/03-05-01.html中。

3.4 表格渲染

如果表格中只能显示文字，那就太单调了。但想让单元格里显示不同颜色的内容，或者干脆显示一张图片，应该怎么做呢？

幸运的是，EXT已经为我们提供了这样的功能。实际上，在上一个示例里我们已经涉及了这样的应用。不同的是，在上一个示例里我们处理的是一个与日期有关的示例。

现在，我们扩充数据data，增加一个表示性别的字段值，如下面的代码所示。

```
var data = [
    ['1','male','name1','descn1'],
    ['2','female','name2','descn2'],
    ['3','male','name3','descn3'],
    ['4','female','name4','descn4'],
    ['5','male','name5','descn5']
];
```

然后修改ds，新增一行配置来描述性别字段，如下面的代码所示。

```
var store = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id'},
        {name: 'sex'},
        {name: 'name'},
        {name: 'descn'}
    ])
});
```

从上面的代码我们可以看到新增了一行{name: 'sex'}，即将数组的第二列映射为性别，这样表格就能知道sex这一列的存在。但是，现在仍然无法显示性别这一列，因为还要修改cm，如下面的代码所示。

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},
    {header:'性别',dataIndex:'sex'},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
]);
```

到目前为止，我们其实只是新增加了一列，与前几个示例并没有太大的区别。现在，我们让不同性别显示不同颜色的字，男士使用红色，女士使用绿色，这样会一目了然。先看一下图3-14显示的效果。

编号	性别	名称	描述
1	红男	name1	descn1
2	绿女	name2	descn2
3	红男	name3	descn3
4	绿女	name4	descn4
5	红男	name5	descn5

图3-14 修改文字颜色

如果大家熟悉HTML和CSS，相信都知道

这是怎么实现的。如果对HTML和CSS不熟悉，建议先了解一下相关的基础知识，因为EXT与HTML和CSS之间的关系非常紧密。JavaScript更需要学习，因为EXT就是轻量级的JavaScript。

修改单元格中内容的颜色的代码如下所示。

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},
    {header:'性别',dataIndex:'sex',renderer:function(value){
            if (value == 'male') {
                return "<span style='color:red;font-weight:bold;'>红男</span>";
            } else {
                return "<span style='color:green;font-weight:bold;'>绿女</span>";
            }
        }},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
]);
```

可以看到，我们在cm里增加了renderer属性，renderer的值是一个自定义函数。不过，这样会让代码显得很乱，所以建议将代码做如下修改。

```
function renderSex(value) {
    if (value == 'male') {
        return "<span style='color:red;font-weight:bold;'>红男</span>";
    } else {
        return "<span style='color:green;font-weight:bold;'>绿女</span>";
    }
}
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},
    {header:'性别',dataIndex:'sex',renderer:renderSex},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
]);
```

大家应该看到了，只需要在返回value之前拼装上相应的HTML和CSS即可。既然可以自定义HTML和CSS，那么在单元格里增加一张图片就是一件很轻松的事了，如图3-15所示。

编号	性别	名称	描述
1	红男	name1	descn1
2	绿女	name2	descn2
3	红男	name3	descn3
4	绿女	name4	descn4
5	红男	name5	descn5

图3-15 显示图片

自定义函数的代码如下所示。

```
function renderSex(value) {
    if (value == 'male') {
        return "<span style='color:red;font-weight:bold;'>红男</span><img src='user_
```

```
            male.png' />";
        } else {
            return "<span style='color:green;font-weight:bold;'>绿女</span><img
src='user_
            female.png' />";
        }
    }
```

如果你对HTML和CSS非常精通，那么一定能轻松实现这些效果。下面我们来尝试一个更复杂的示例，代码如下所示。

```
function renderDescn(value, cellmeta, record, rowIndex, columnIndex, store) {
    var str = "<input type='button' value='查看详细信息' onclick='alert(\"" +
        "这个单元格的值是：" + value + "\\n" +
        "这个单元格的配置是：{cellId:" + cellmeta.cellId + ",id:" + cellmeta.id + ",css:"
        + cellmeta.css + "}\\n" +
        "这个单元格对应行的record是：" + record + "，一行的数据都在里边\\n" +
        "这是第" + rowIndex + "行\\n" +
        "这是第" + columnIndex + "列\\n" +
        "这个表格对应的Ext.data.Store在这里：" + store + "，随便用吧。" +
        "\")'>";
    return str;
}
```

我们可以在renderer里得到多个参数，如下所示。

- value：将要显示到单元格里的值。
- cellmeta：单元格的相关属性，主要有id和CSS。
- record：这行的数据对象，如果需要获取其他列的值，可以通过record.data["id"]的方式得到。
- rowIndex：行号，这里的行号指的是当前页中所有记录的顺序。
- columnIndex：当前列的列号。
- store：构造表格时传递的ds。也就是说，表格里的所有数据都可以通过store获得。

大家肯定有疑问，我们只是将一个自定义函数设置到renderer上，这些参数是从哪里来的呢？

其实这个问题比较简单，既然是函数，就肯定有调用它的地方，我们只要打开EXT源码GridView.js找到调用renderer的地方，就可以看到这些参数是如何传递到我们的自定义函数中的。

看看效果，如图3-16所示。



图3-16 复杂渲染效果

完整代码如下所示：

```
function renderSex(value) {
    if (value == 'male') {
        return "<span style='color:red;font-weight:bold;'>红男</span><img src='user
           _male.png' />";
    } else {
        return "<span style='color:green;font-weight:bold;'>绿女</span><img src=
            'user_female.png' />";
    }
}

function renderDescn(value, cellmeta, record, rowIndex, columnIndex, store) {
    var str = "<input type='button' value='查看详细信息' onclick='alert(\"" +
        "这个单元格的值是：" + value + "\\n" +
        "这个单元格的配置是：{cellId:" + cellmeta.cellId + ",id:" + cellmeta.id
        + ",css:" + cellmeta.css + "}\\n" +
        "这个单元格对应行的record是：" + record + "，一行的数据都在里边\\n" +
        "这是第" + rowIndex + "行\\n" +
        "这是第" + columnIndex + "列\\n" +
        "这个表格对应的Ext.data.Store在这里：" + store + "，随便用吧。" +
        "\")'>";
    return str;
}

var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},
    {header:'性别',dataIndex:'sex',renderer:renderSex},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn',renderer:renderDescn}
]);

var data = [
    ['1','male','name1','descn1'],
    ['2','female','name2','descn2'],
    ['3','male','name3','descn3'],
    ['4','female','name4','descn4'],
    ['5','male','name5','descn5']
];

var store = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id'},
        {name: 'sex'},
        {name: 'name'},
        {name: 'descn'}
    ])
});
store.load();

var grid = new Ext.grid.GridPanel({
    autoHeight: true,
    renderTo: 'grid',
```

```
        store: store,
        cm: cm
});
```

该示例在03.grid目录下的04-01.html中。

3.5 给表格的行和列设置颜色

即便是带有斑马条的表格也难以满足追求更炫丽的页面的客户的需求。所以，我们可以设置表格的行或列的颜色，从而产生更好的视觉效果，给行设置颜色的代码如下所示。

```
viewConfig : {
  forceFit : true,
       enableRowBody : true,
       getRowClass :function(record, rowIndex, p, ds){
         var cls = 'white-row';
         switch (record.data.color){
           case '#FBF8BF' :
                        cls = 'yellow-row'
                        break;
           case '#99CC99' :
                        cls = 'green-row'
                        break;
            case '#F5C0C0' :
                        cls = 'red-row'
                        break;
          }
         return cls;
       }
}
```

CSS中相应的代码如下所示。

```
#uses the following css:
 .red-row{ background-color: #F5C0C0 !important; }
 .yellow-row{ background-color: #FBF8BF !important; }
 .green-row{ background-color: #99CC99 !important; }
```

运行结果如图3-17所示。

name	sex
boy	0
girl	1
man	0
woman	1

图3-17　自定义行的颜色

下面的代码用来动态修改列的背景色：

```
function renderMotif(data, cell, record, rowIndex, columnIndex, store){
    var value = record.get('color')
```

```
        cell.attr = "style=background-color:" + value;
        return data;
    }

    var cm = new Ext.grid.ColumnModel([
        {header: 'name', dataIndex: 'name'},
        {header: 'sex', dataInex: 'sex'},
        {header: 'color', dataIndex: 'color', renderer: renderMotif}
    ]);
```

运行结果如图3-18所示。

name	sex	color
boy	0	#FBF8BF
girl	1	#FBF8BF
man	0	#FBF8BF
woman	1	#FBF8BF

图3-18 自定义列的颜色

当然，如果只修改表格的某一行的样式，还可以用下面的方式实现。

```
grid.getView().addRowClass(r,css)
```

修改表格某一单元格的样式可以用下面的方式实现。

```
Ext.get(grid.getView().getCell(r,c)).addClass(css)
```

或

```
grid.getView().getRow(r).style.backgroundColor="red";
```

3.6 自动显示行号和复选框

实际上，行号和复选框都是renderer的延伸。当然，复选框的功能要复杂得多，两者经常一起使用，所以我们放在一起讲述。

3.6.1 自动显示行号

修改前面示例中的列模型cm，加入RowNumberer对象，如下面的代码所示。

```
var cm = new Ext.grid.ColumnModel([
    new Ext.grid.RowNumberer(),
    {header:'编号',dataIndex:'id'},
    {header:'性别',dataIndex:'sex'},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
]);
```

从图3-19中可以看到，表格最左边自动显示了行号。

该示例在03.grid/06-01-01.html中。

关于自动计算行号，它总伴随着一个小小的问题。如果删除表格中间的一行，那么行号就不连续了，所以需要刷新表格的视图，让表格重新计算行号。

	编号	性别	名称	描述
1	1	male	name1	descn1
2	2	female	name2	descn2
3	3	male	name3	descn3
4	4	female	name4	descn4
5	5	male	name5	descn5

图3-19 自动显示行号

我们在HTML里添加一个按钮，将它的id设置为remove，然后在按下按钮时删除表格的第2行数据，代码如下所示。

```
Ext.get('remove').on('click', function() {
    store.remove(store.getAt(1));
});
```

我们会在浏览器中发现页面里多了一个按钮，上面的代码中使用Ext.get()获得这个按钮，并监听它的click事件，执行ds.remove(ds.getAt(1))，ds.getAt方法用于获得某一行数据，删除后的效果如图3-20所示。

	编号	性别	名称	描述
1	1	male	name1	descn1
3	3	male	name3	descn3
4	4	female	name4	descn4
5	5	male	name5	descn5

删除第二行

图3-20 删除后行号不能连续

可以看到删除第2行数据后，行号变得不连续了。再来看看下面的这段代码：

```
Ext.get('remove').on('click', function() {
    store.remove(store.getAt(1));
    grid.view.refresh();
});
```

增加了一行代码，用于在执行删除操作后刷新表格视图。刷新后，表格就会调用renderer()重新计算所有行号。这样便得到我们想要显示的效果了。完整代码如下所示：

```
var cm = new Ext.grid.ColumnModel([
    new Ext.grid.RowNumberer(),
    {header:'编号',dataIndex:'id'},
    {header:'性别',dataIndex:'sex'},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
]);

var data = [
```

```
        ['1','male','name1','descn1'],
        ['2','female','name2','descn2'],
        ['3','male','name3','descn3'],
        ['4','female','name4','descn4'],
        ['5','male','name5','descn5']
    ];

    var store = new Ext.data.Store({
        proxy: new Ext.data.MemoryProxy(data),
        reader: new Ext.data.ArrayReader({}, [
            {name: 'id'},
            {name: 'sex'},
            {name: 'name'},
            {name: 'descn'}
        ])
    });
    store.load();

    var grid = new Ext.grid.GridPanel({
        autoHeight: true,
        renderTo: 'grid',
        store: store,
        cm: cm
    });

    Ext.get('remove').on('click', function() {
        store.remove(store.getAt(1));
        grid.view.refresh();
    });
```

最终页面显示效果如图3-21所示。

	编号	性别	名称	描述
1	1	male	name1	descn1
2	3	male	name3	descn3
3	4	female	name4	descn4
4	5	male	name5	descn5

删除第二行

图3-21　刷新自动行号

该示例在03.grid/06-01-02.html中。

3.6.2　复选框

为了在表格中添加复选功能，我们需要使用CheckboxSelectionModel，它会在每行数据前添加一个复选框。同样，我们这里还是要修改cm，SelectionModel对象即sm，它在总体上控制用户对表格的选择功能。以前这类多选选择功能都可以利用Shift或Ctrl键实现，现在都要与复选框关联上了。

首先我们要创建一个sm，也就是CheckboxSelectionModel()。sm身兼两职，使用时既要放到cm里，也要放到表格中，如下面的代码所示。

```
var sm = new Ext.grid.CheckboxSelectionModel();
var cm = new Ext.grid.ColumnModel([
    new Ext.grid.RowNumberer(),
    sm,
    {header:'编号',dataIndex:'id'},
    {header:'性别',dataIndex:'sex'},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
]);
var grid = new Ext.grid.GridPanel({
    renderTo: 'grid',
    store: store,
    cm: cm,
    sm: sm
});
```

显示效果如图3-22所示。

该示例在03.grid/06-02-01.html中。

注意 虽然CheckboxModelSection允许我们使用复选框选中表格里的多行，但是如果在操作过程中不慎选中了某一行，就会变成选中一行的情况。有没有办法取消原始的选择功能，只允许用户通过复选框执行选中操作呢？可以，只需要重设行选择事件的处理函数，如下所示：

```
var sm = new Ext.grid.CheckboxSelectionModel({handleMouseDown: Ext.emptyFn});
```

	☑	编号	性别	名称	描述
1	☑	1	male	name1	descn1
2	☑	2	female	name2	descn2
3	☑	3	male	name3	descn3
4	☑	4	female	name4	descn4
5	☑	5	male	name5	descn5

图3-22 复选框多选行

3.7 选择模型

表格里提供的这种功能称为选择模型。例如，当单击某一个单元格时，被选中的却是整个行。

在定义Ext.grid.GridPanel时，默认使用RowSelectionModel——行选择模型。行选择模型是默认支持多选的，鼠标单击时按住Shift或Ctrl键就可以选择多行。如果只希望选择一行，需要设置singleSelect参数。

如下面的代码所示：

```
var grid = new Ext.grid.GridPanel({
    renderTo: 'grid',
    store: store,
    cm: cm,
    sm: new Ext.grid.RowSelectionModel({singleSelect:true})
});
```

这样，即使按着Ctrl或者Shift键也不会选中多行了，只会突出显示最后选中的一行。

该示例在03.grid/07-01.html中。

另外一种选择模型是CellSelectionModel——单元格选择模型。每次只允许选择一个单元格，在EditorGrid里默认使用的就是CellSelectionModel。当然，也可以将EditorGrid的选择模型设置为RowSelectionModel，从而达到选中一行的效果。

可能大家会问："选择模型具体有什么用？我还是不知道如何读取选中的行呀。"例如，当单击某一行时就会弹出一个对话框，并且显示当前行的一些信息。

这就需要用到选择模型了，如下面的代码所示。

```
grid.on('click', function() {
    var selections = grid.getSelectionModel().getSelections();
    for (var i = 0; i < selections.length; i++) {
        var record = selections[i];
        Ext.Msg.alert('提示', record.get("id") + "," + record.get("name") + "," +
        record.get("descn"));
    }
});
```

先从表格里获得SelectionModel，再从选择模型中获得当前选中的数据，selections.length是选中的记录条数。循环读取selections，里面的每一个元素都是一条记录，所有的数据都在它里面。如果需要判断是否选择了记录，只需要判断selections.length是否等于0即可。

该示例在03.grid/07-02.html中。

注意 下面说一些题外话，EXT的树形组件里也有两种选择模型，默认的DefaultSelection-Model每次只能选择一个节点，另外还有一个MultiSelectionModel，它可以使用Ctrl键（注意，不允许使用Shift键）选择多个节点。

3.8 表格视图——Ext.grid.GridView

EXT的表格控件都遵守MVC模型，Ext.data.Store可看作模型（Model），Ext.grid.GridPanel中设置的各种监听器可看作控制器（Controller），而Ext.grid.GridView对应的就是视图（View）。通常情况下，不需要自行创建Ext.grid.GridView的实例，Ext.grid.GridPanel会自动生成对应的实例，使用默认的样式将表格显示到页面上。当希望操作Ext.grid.GridView的属性时，可以通过Ext.grid.GridPanel的getView()函数来获取当前表格使用的视图实例，如下面代码所示。

```
Ext.get('scroll').on('click', function() {
    grid.getView().scrollToTop();
});
Ext.get('focus').on('click', function() {
    grid.getView().focusCell(0, 0);
    var cell  = grid.getView().getCell(0, 0);
    cell.style.backgroundColor = 'red';
});
```

我们设置了两个按钮“scroll to top”和“focus cell”。

- 单击“scroll to top”按钮，会调用`grid.getView().scrollToTop()`函数。该函数的作用是，当GridView的右侧显示滑动条时，自动将滑动条滚动到最上面的位置。
- 单击“focus cell”按钮，首先调用`focusCell(0,0)`将焦点放在表格的第一行的第一个单元格上，然后将这个单元格的背景设置为红色。

上面示例显示的效果如图3-23所示。

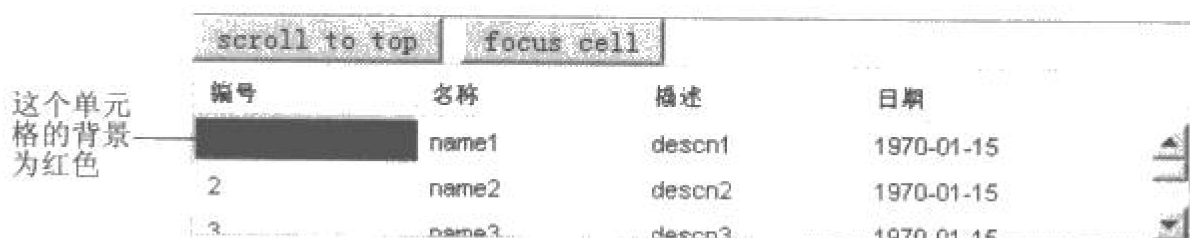


图3-23 GridView滚动到顶端并选中某个单元格

从图3-23中可以看到，与GridView相关的操作都集中在视图的显示功能部分。如果要对表格的显示效果进行调整，可以通过GridView进行设置，但是`grid.getView()`必须在创建`Ext.grid.GridPanel`之后调用，它只能获得`Ext.grid.GridPanel`中创建好的GridView实例。如果我们希望在创建GridView时设置一些初始参数，可以使用`Ext.grid.GridPanel`的`viewConfig`参数，如下面的代码所示。

```
var grid = new Ext.grid.GridPanel({
    height: 80,
    width: 450,
    renderTo: 'grid',
    store: new Ext.data.Store({
        autoLoad: true,
        proxy: new Ext.data.MemoryProxy(data),
        reader: new Ext.data.ArrayReader({}, meta)
    }),
    columns: meta,
    viewConfig: {
        columnsText: '显示的列',
        scrollOffset: 30,
        sortAscText: '升序',
        sortDescText: '升序',
        forceFit: true
    }
});
```

viewConfig中的参数会在GridView创建时作为初始化参数传递给GridView，上例中列举了日常开发中常用的一些参数，如下所示。

- columnsText、sortAscText、sortDescText这3个参数分别用来设置表格中每列的下拉菜单中的“显示的列”、“升序”、“降序”这3个部分显示的文字。
- scrollOffset表示表格右侧为滚动条预留的宽度，默认是20 px。
- forceFit参数为true时，表格会自动延展每列的长度，使内容填满整个表格。

上例的页面效果如图3-24所示。



图3-24 使用viewConfig为GridView设置参数

EXT中基于GridView创建了可以提供更多扩展功能的视图组件，比较常用的有分组视图GroupingView和可以支持数据缓冲的BufferedGridView。有关GroupingView的介绍在3.13节中，BufferedGridView作为Ext 3.0扩展的一部分包含在发布包中，有关它的介绍在第14章中。

3.9 表格分页

一次性将成千上万条数据显示在表格里，然后拖动滚动条查看数据，显然不是什么好主意，在效率上也是不允许的。

实际上，表格控件对性能的要求较高。在一个页面上放3个表格，就可以感觉到响应变慢。在每个表格里显示上千条数据，效率就可想而知了。

所以说分页是必不可少的，而EXT提供了方便地集成分页工具条的方式。

3.9.1 为表格添加分页工具条

在先前示例中的表格上添加几行代码，如下所示。

```
var grid = new Ext.grid.GridPanel({
    renderTo: 'grid',
    autoHeight: true,
    store: store,
    cm: cm,
    bbar: new Ext.PagingToolbar({
        pageSize: 10,
        store: store,
        displayInfo: true,
        displayMsg: '显示第 {0} 条到 {1} 条记录，一共 {2} 条',
        emptyMsg: "没有记录"
```

```
    })
});
store.load();
```

我们新增了一个bbar属性，并且创建了Ext.PagingToolbar分页工具条对象。

分页工具条的属性如下所示。

- pageSize：每页显示几条数据。
- displayInfo：是否显示数据信息。
- displayMsg：只有当displayInfo:true时才有效，用来显示有数据时的提示信息，{0}、{1}、{2}会自动被替换成对应的数据。
- emptyMsg：没有数据时显示的信息。

不过要注意一点，如果配置了分页工具条，store.load()就必须在构造表格以后执行，否则分页工具条将不起作用。应该将分页工具条和ds相关联，从而实现与表格共享数据模型，分页后的效果如图3-25所示。

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

Page 1 of 1　显示第 1 条到 5 条记录，一共 5 条

图3-25　分页工具条

从图3-25可以清晰地看到，在表格下边多出来一个工具条，包括前一页、后一页、第一页、最后一页、刷新以及提示信息。

该示例在03.grid/09-01-01.html中。

曾经有一位开发者在使用过程中提出了一个挺奇怪的需求——让pagingToolbar右对齐。

这应该是非常难碰到的需求，EXT中没有直接提供配置的参数。尝试将float:right放到CSS中，让div移到右边，但结果是style="float:right"没有效果，只好使用cls指定一个定义好的class。

EXT 3.0中提供了更多效果绚丽的分页组件，可以让用户使用滑动条或者进度条实现分页的效果，这些扩展组件可以在第14章中找到，本节示例在03.grid/09-01-02.html中。

注意　此处的ds不能使用Ext.data.SimpleStore，分页时需要调用store的load()函数，而load()函数与Proxy有关。SimpleStore不但没有设置Proxy，而且也没有重写load()函数，所以会出现错误，从而导致无法显示分页信息。

3.9.2　通过后台脚本获得分页数据

表格每次都会显示ds中的所有数据，我们无法利用静态数据很好地演示分页。于是，必须写

后台脚本，让EXT与后台进行数据交互，这样才能看到真实的分页效果。

尽量在原来示例的基础上修改，把注意力集中在关键部分。

使用JSP来写后台代码。至于后台语言，不论是ASP还是JSP，只要返回的数据格式符合要求即可，即Ajax与后台无关。也就是说，不管用什么语言编写后台代码，前台的EXT代码都一样，如代码清单3-2所示。

代码清单3-2　用JSP编写实现分页功能的后台代码

```
<%
String start = request.getParameter("start");
String limit = request.getParameter("limit");
try {
    int index = Integer.parseInt(start);
    int pageSize = Integer.parseInt(limit);

    String json = "{totalProperty:100,root:[";
    for (int i = index; i < pageSize + index; i++) {
        json += "{id:" + i + ",name:'name" + i + "',descn:'descn" + i + "'}";
        if (i != pageSize + index - 1) {
            json += ",";
        }
    }
    json += "]}";
    response.getWriter().write(json);
} catch(Exception ex) {
}
%>
```

下面我们来解读这段JSP代码。

- 在进行操作之前，先要获得EXT传递过来的两个参数：`start`和`limit`。`start`指示从第几个数据开始显示；`limit`指从`start`开始一共要用多少条数据。当然，返回的数据可能会小于这个值。
- 在后台模拟对100条数据进行分页，在获得了`start`和`limit`之后生成JSON格式的数据。

什么是JSON？在讲解理论之前先看看示例，建立一下感性认识。

模拟EXT访问后台，并传递两个参数`start=0&limit=10`，把获得的数据稍微整理一下，如下所示。

```
{totalProperty:100,root:[
    {id:0,name:'name0',descn:'descn0'},
    {id:1,name:'name1',descn:'descn1'},
    {id:2,name:'name2',descn:'descn2'},
    {id:3,name:'name3',descn:'descn3'},
    {id:4,name:'name4',descn:'descn4'},
    {id:5,name:'name5',descn:'descn5'},
    {id:6,name:'name6',descn:'descn6'},
    {id:7,name:'name7',descn:'descn7'},
    {id:8,name:'name8',descn:'descn8'},
    {id:9,name:'name9',descn:'descn9'}
]}
```

请记住这个数据格式，不管后台是什么，只要满足了这样的格式要求，EXT就可以接受并处理，然后显示到表格中。

这里先不介绍JSON，只需要知道JSON里除了name（名称）就是value（值）。值有好几种格式，如果是数字，不用加引号；如果加了引号，就是字符串；如果用[]包裹，就是数组；如果出现{}，就说明是嵌套的JSON，诸如此类。

简单看一下JSON数据，开头就是totalProperty:100，这里表示一共有100条数据。然后就是root:[]，root对应着一个数组，数组里有10个对象，每个对象都有id、name和descn。这10条数据最后就应该显示到表格里。

3

JSP用for循环生成root数组里的数据，这样翻页时就可以看到数据的变化。否则每次都是一样的数据，我们不知道翻页是否起了作用。最后，把得到的JSON字符串输出到response里，EXT就可以获得这些数据了。

现在已经确定JSP可以返回我们所需要的数据了，可以不管后台是用什么语言写的，直接切入EXT代码，看看它是如何与后台交互并获得这些数据的。

因为引入了JSON作为数据传输格式，所以这次要对前几个示例的代码进行一次大的修改，具体步骤如下。

(1) 换掉Proxy，不再到内存中查找，而是通过HTTP获得我们想要的数据。

```
proxy: new Ext.data.HttpProxy({url:'08_02_01.jsp'}),
```


创建HttpProxy的同时，用url这个参数指定获取数据的路径，我们这里设置成09_02_01.jsp，也就是我们刚才讨论的JSP脚本。

(2) 现在不再是解析简单的数组，而是换成JsonReader，如下面的代码所示。

```
reader: new Ext.data.JsonReader({
    totalProperty: 'totalProperty',
    root: 'root'
}, [
    {name: 'id'},
    {name: 'name'},
    {name: 'descn'}
])
```

注意，这里比ArrayReader多了什么？totalProperty对应JSP代码中返回的totalProperty，也就是数据的总数。root对应JSP代码中返回的root，也就是一个包含返回数据的数组。

(3) 最后，在初始化时通过传参数去获得希望得到的那部分的数据，如下面的代码所示。

```
store.load({params:{start:0,limit:10}});
```

上面代码中，在Store进行加载时额处传递了两个参数（start和limit），这是告诉后台程序从第1条数据开始，最多读取10条。

不过，如果按照我们以前的设置，表格是无法正常显示的。因为store.load()无法在grid.render()前准备好所有数组，所以它不知道应该显示多高。我们需要为表格指定一个固定的高度，如<div id="grid" style="height:265px;"></div>，或者为它添加一个autoHeight:true参数，让它自己计算高度。

最后，我们就可以使用分页条上那些按钮，试试分页的功能。前台JavaScript脚本内容如下所示：

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
]);

var store = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url:'09_02_01.jsp'}),
    reader: new Ext.data.JsonReader({
        totalProperty: 'totalProperty',
        root: 'root'
    }, [
        {name: 'id'},
        {name: 'name'},
        {name: 'descn'}
    ])
});

var grid = new Ext.grid.GridPanel({
    renderTo: 'grid',
    autoHeight: true,
    store: store,
    cm: cm,
    bbar: new Ext.PagingToolbar({
        pageSize: 10,
        store: store,
        displayInfo: true,
        displayMsg: '显示第 {0} 条到 {1} 条记录，一共 {2} 条',
        emptyMsg: "没有记录"
    })
});
store.load({params:{start:0,limit:10}});
```

该示例在03.grid/09-02-01.html中[①]，JSP文件在03.grid/09_02_01.jsp中，浏览效果如图3-26所示。

编号	名称	描述
40	name40	descn40
41	name41	descn41
42	name42	descn42
43	name43	descn43
44	name44	descn44
45	name45	descn45
46	name46	descn46
47	name47	descn47
48	name48	descn48
49	name49	descn49

Page 5 of 10 显示第 41 条到 50 条记录，一共 100 条

图3-26 使用后台数据分页

① 此实例需要后台服务器的支持，可以在光盘的apache-tomcat-5.5.28\webapps\ext-3.0.0\examples\03.grid目录下找到该实例的HTML和JSP页面。

3.9.3 分页工具条显示在表格的顶部

除了bbar以外，表格还有tbar，即上方的工具条。如果把分页条放在上面，效果也是一样的。也就是将原来的bbar（bottom bar）属性修改为tbar（top bar），便可以像图3-27那样将工具条显示到表格的上方了，该示例在03.grid/09-03-01.html中。

```
var grid = new Ext.grid.GridPanel({
    renderTo: 'grid',
    store: store,
    cm: cm,
    tbar: new Ext.PagingToolbar({
        pageSize: 10,
        store: store,
        displayInfo: true,
        displayMsg: '显示第 {0} 条到 {1} 条记录，一共 {2} 条',
        emptyMsg: "没有记录"
    })
});
```

Page 1 of 1　显示第 1 条到 5 条记录，一共 5 条

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

图3-27　顶端放置分页工具条

当然，也可以在上下都加上分页条，因为它们都共享同一个store，在功能上不会有任何问题。

3.9.4 让 EXT 支持前台分页

前台分页一次性从后台把所有数据都读取到客户端，然后由客户端自动判断每次显示多少条数据。这样，分页时就不用再去后台读取数据了。这对于小数据量的分页是非常有利的。

不过，EXT里并没有提供这样的功能，表格只是把得到的所有数据一次性显示到表格里，无论pageSize设置为多少都不会起作用。

虽然EXT并没有直接为我们提供这样的内存分页功能，但是在EXT2.0的examples/locale/目录下提供了一个PagingMemoryProxy.js的扩展，它可以让我们从本地数组读取数据，并且实现内存分页，具体使用步骤如下所示。

(1) 将PagingMemoryProxy.js从examples/locale/目录下复制过来，导入到HTML里。

```
<script type="text/javascript" src="PagingMemoryProxy.js"></script>
```

(2) 把以前的MemoryProxy换成PagingMemoryProxy。

```
var store = new Ext.data.Store({
    proxy: new Ext.data.PagingMemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, [
```

```
            {name: 'id'},
            {name: 'name'},
            {name: 'descn'}
        ])
    });
```

(3) 好了，现在直接调用ds.load({params:{start:0,limit:3}})，把最前面的3条记录显示出来吧，如图3-28所示。

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3

Page 1 of 2 显示第1条到3条记录，一共5条

图3-28 前台分页

(4) 可以在前台生成JavaScript数组，或者使用Ajax读取后台数据，再传递给PagingMemoryProxy，以此实现内存分页的功能。

该示例在03.grid/09-04-01.html中。

3.10 后台排序

默认情况下，表格只能对当前页的数据进行排序。如果想要对所有数据进行排序，则需要把排序信息提交到后台，由后台将排序信息组装到SQL里，然后再由后台将处理好的数据返回给前台。

这就是前台与后台交互的过程，前台只是将后台处理好的数据显示到表格里，后台只需要返回格式正确的数据。这些内容在上一节中已经讲到，这里只是在分页的基础上加上了排序。

既然要提交到服务端，便需要将Ext.data.Store的remoteSort属性设置为true，这个属性是指是否允许远程排序，默认值为false。实现后台排序的过程如代码清单3-3所示。

代码清单3-3 后台排序

```
var store = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url:'10_01.jsp'}),
    reader: new Ext.data.JsonReader({
        totalProperty: 'totalProperty',
        root: 'root'
    }, [
        {name: 'id'},
        {name: 'name'},
        {name: 'descn'}
    ]),
    remoteSort: true
});
```

将这个属性值设置为true后，下次排序时就会有变化，不会立即显示出排序的结果，而是向后台提交了两个参数，分别是sort和dir。sort表示需要排序的字段，dir表示升序或降序

(ASC/DESC)。然后，后台根据这些参数对数据进行处理。

该示例在10_01.jsp中。

使用JSP编写的后台排序代码如代码清单3-4所示。

代码清单3-4 实现后台排序的JSP代码

```
<%
String start = request.getParameter("start");
String limit = request.getParameter("limit");

String sort = request.getParameter("sort");
String dir = request.getParameter("dir");
System.out.println(dir);
if (dir == null) {
    dir = "ASC";
}

try {
    int index = Integer.parseInt(start);
    int pageSize = Integer.parseInt(limit);

    String json = "{totalProperty:100,root:[";

    if (dir.equals("ASC")) {
        for (int i = index; i < pageSize + index; i++) {
            json += "{id:" + i + ",name:'name" + i + "',descn:'descn" + i + "'}";
            if (i != pageSize + index - 1) {
                json += ",";
            }
        }
    } else {
        for (int i = pageSize + index; i > index; i--) {
            json += "{id:" + i + ",name:'name" + i + "',descn:'descn" + i + "'}";
            if (i != index - 1) {
                json += ",";
            }
        }
    }
    json += "]}";
    response.getWriter().write(json);
} catch(Exception ex) {
}
%>
```

从以上代码中可以看出，使用request.getParameter()就可以得到那两个参数了。我们这里没有连接数据库，只对dir进行了处理。如果是null或ASC，返回的数据就按升序排列；如果是DESC，返回的数据就按降序排列，结果如图3-29所示。

编号	名称	描述
0	name0	descn0
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5
6	name6	descn6
7	name7	descn7
8	name8	descn8
9	name9	descn9

Page 1 of 10

图3-29 后台排序

在JSP代码中，根据排序的方式执行i++或i--操作。如果用SQL，那么就会更简单。直接用这两个字段就可以编写出需要的SQL语句，如下面的代码所示。

```
String sql = "select * from t_user order by " + sort + " " + dir;
```

其实它就是为SQL而准备的。

该示例在03.grid/10-01.html中。

3.11 可编辑表格控件——EditorGrid

相信大家都使用过Microsoft Excel，它的功能很强大，用户可随意添加或删除表格中的行和列，而且只保存一次即可。EditorGrid也提供了这些功能，可以直接在表格里执行添加、删除、修改和查找等操作，然后一次性保存。

还可以动态修改某个单元格，这些单元格我们先暂定不能为空，保存时会进行检测，为空就无法保存，验证信息会给予提示。

3.11.1 制作一个简单的 EditorGrid

首先，同样是定义列，如下面的代码所示。

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id',editor:new Ext.grid.GridEditor(new Ext.form.
    TextField({
        allowBlank: false
    }))},
    {header:'名称',dataIndex:'name',editor:new Ext.grid.GridEditor(new Ext.form.
    TextField({
        allowBlank: false
    }))},
    {header:'描述',dataIndex:'descn',editor:new Ext.grid.GridEditor(new Ext.form.
    TextField({
        allowBlank: false
    }))}
]);
```

大家可以看到，我们现在给每列增加editor属性，里边的属性都是完全一样的TextField。假设它就是用来编辑单元格的，接着往下看。

```
var grid = new Ext.grid.EditorGridPanel({
    renderTo: 'grid',
    store: store,
    cm: cm
});
```

在这里，我们发现除了在GridPanel的前面多了个Editor外，其他的部分并没有什么变化。可是看到的结果是，现在可以用TextField的方式随意修改单元格。记得不能让单元格为空，否则无法修改。

默认情况下，需要双击单元格才能激活编辑器，从而进行修改。不过，也可以给表格配置上clicksToEdit:1，这样就可以通过单击单元格激活编辑器，从而进行修改，如图3-30所示。

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

图3-30 通过单击修改单元格

TextField不允许输入空值，因为在创建ColumnModel时为对应的editor设置了allowBlank:false属性，allowBlank:false表示不允许在TextField中输入空值。

完整代码如下所示：

```
var cm = new Ext.grid.ColumnModel([{
    header: '编号',
    dataIndex: 'id',
    editor: new Ext.grid.GridEditor(
        new Ext.form.TextField({
            allowBlank: false
        })
    )
}, {
    header: '名称',
    dataIndex: 'name',
    editor: new Ext.grid.GridEditor(
        new Ext.form.TextField({
            allowBlank: false
        })
    )
}, {
    header: '描述',
    dataIndex: 'descn',
    editor: new Ext.grid.GridEditor(
        new Ext.form.TextField({
            allowBlank: false
        })
    )
}]);

var data = [
    ['1','name1','descn1'],
    ['2','name2','descn2'],
    ['3','name3','descn3'],
    ['4','name4','descn4'],
    ['5','name5','descn5']
];
var store = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id'},
        {name: 'name'},
        {name: 'descn'}
```

```
    ])
});

store.load();
var grid = new Ext.grid.EditorGridPanel({
    autoHeight: true,
    renderTo: 'grid',
    store: store,
    cm: cm
});
```

该示例在03.grid/11-01-01.html中。

3.11.2 添加一行数据

对可编辑的表格来说，添加、删除、修改和查找是最基本的功能，不然编辑就失去了意义。

首先使用Record的create方法创建一个记录集MyRecord。MyRecord其实相当于一个类，该类包含了记录集的定义信息，可以通过MyRecord来创建包含字段的Record对象，如下面的代码所示。

```
var MyRecord = Ext.data.Record.create([
    {name: 'id', type: 'string'},
    {name: 'name', type: 'string'},
    {name: 'descn', type: 'string'}
]);
```

现在我们要两个按钮，一个可以添加行，另一个执行反向操作（删除行），暂且就把它们放到表格的首部。

在代码清单3-5中，我们可以通过MyRecord.getField("name")得到记录中name列的字段信息，通过p.get("name")可以得到记录name字段的值，而通过p.data.name同样能得到记录集中name字段的值。如果对Record有一定的了解，那么要操作记录集中的数据就非常简单了，比如p.set(name,value)可以设置记录中某指定字段的值，p.dirty可以判断当前记录是否有字段的值被更改过，等等。

代码清单3-5 添加与删除行

```
var grid = new Ext.grid.EditorGridPanel({
    renderTo: 'grid',
    store: store,
    cm: cm,
    tbar: new Ext.Toolbar(['-', {
        text: '添加一行',
        handler: function(){
            var p = new MyRecord({
                id:'',
                name:'',
                descn:''
            });
            grid.stopEditing();
            store.insert(0, p);
```

```
                grid.startEditing(0, 0);
            }
        }, '-', {
            text: '删除一行',
            handler: function(){
                Ext.Msg.confirm('信息', '确定要删除? ', function(btn){
                    if (btn == 'yes') {
                        var sm = grid.getSelectionModel();
                        var cell = sm.getSelectedCell();
                        var record = store.getAt(cell[0]);
                        store.remove(record);
                    }
                });
            }
        }, '-'])
    });
```

在上面的代码中，我们通过tbar创建一个工具条，然后在这个工具条里放两个按钮：一个叫“添加一行”，另一个叫“删除一行”。它们用text定义按钮显示的文字，handler定义按钮被按下时执行的函数。

我们来看看“添加一行”按钮里执行的函数：首先新建一个MyRecord（记得给里面的属性赋值，否则EditorGrid最后显示的内容就会混乱；然后，关闭表格的编辑状态，再把我们刚才创建的MyRecord插入store的第一行。数据模型一旦改变，表格也会立即改变。startingEditing()激活第1行第1列的编辑状态，提示你最好现在就开始写数据。dirty和modifed的操作是为了表格对脏数据部分特殊显示，也就是上面所说的数据是否被更改过。

示例中的删除函数先获得表格的选择模型，从选择模型中获得选中的单元格。这个单元格有两个属性，第一个是行号，第二个是列号。我们通过行号得到store这一行对应的Record，然后移除即可。

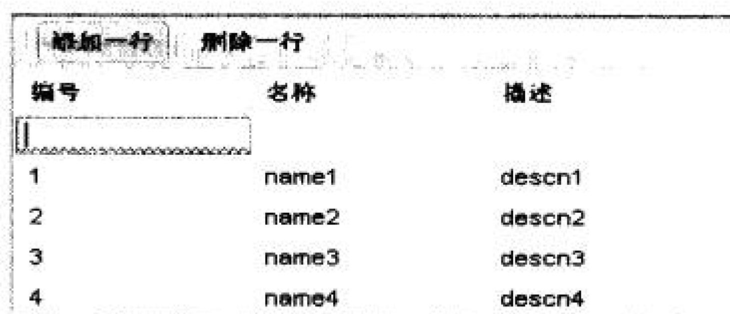


图3-31　添加和删除按钮

就这样，我们完成了添加和删除的操作，并且尝试了工具条的用法。另外，请注意，两个按钮之间的 '-' 是分隔符，适当使用可以使工具条中的按钮显得排列有序，如图3-31所示。

完整代码如下所示：

```
var cm = new Ext.grid.ColumnModel([{
    header: '编号',
    dataIndex: 'id',
    editor: new Ext.grid.GridEditor(
        new Ext.form.TextField({
            allowBlank: false
        })
    )
}, {
    header: '名称',
    dataIndex: 'name',
```

```
        editor: new Ext.grid.GridEditor(
            new Ext.form.TextField({
                allowBlank: false
            })
        )
    }, {
        header: '描述',
        dataIndex: 'descn',
        editor: new Ext.grid.GridEditor(
            new Ext.form.TextField({
                allowBlank: false
            })
        )
    }]);

    var data = [
        ['1','name1','descn1'],
        ['2','name2','descn2'],
        ['3','name3','descn3'],
        ['4','name4','descn4'],
        ['5','name5','descn5']
    ];

    var store = new Ext.data.Store({
        proxy: new Ext.data.MemoryProxy(data),
        reader: new Ext.data.ArrayReader({}, [
            {name: 'id'},
            {name: 'name'},
            {name: 'descn'}
        ])
    });

    var Record = Ext.data.Record.create([
        {name: 'id', type: 'string'},
        {name: 'name', type: 'string'},
        {name: 'descn', type: 'string'}
    ]);
    store.load();

    var grid = new Ext.grid.EditorGridPanel({
        autoHeight: true,
        renderTo: 'grid',
        store: store,
        cm: cm,
        tbar: new Ext.Toolbar(['-', {
            text: '添加一行',
            handler: function(){
                var p = new Record({
                    id:'',
                    name:'',
                    descn:''
                });
                grid.stopEditing();
                store.insert(0, p);
```

```
                grid.startEditing(0, 0);
            }
        }, '-', {
            text: '删除一行',
            handler: function(){
                Ext.Msg.confirm('信息', '确定要删除？', function(btn){
                    if (btn == 'yes') {
                        var sm = grid.getSelectionModel();
                        var cell = sm.getSelectedCell();

                        var record = store.getAt(cell[0]);
                        store.remove(record);
                    }
                });
            }
        }, '-'])
});
```

该示例在03.grid/11-02-01.html中。

3.11.3 保存修改结果

大家注意，必须使用上节中介绍的添加行的方式，才能保证从store.modified获得新增的行，否则可能出现校验错误。

首先在工具条上添加一个保存按钮，实现方法如代码清单3-6所示。

代码清单3-6 为工具条添加保存按钮

```
{
    text: '保存',
    handler: function(){
        var m = store.modified.slice(0);
        var jsonArray = [];
        Ext.each(m, function(item) {
            jsonArray.push(item.data);
        });

        Ext.lib.Ajax.request(
            'POST',
            '11_03_01.jsp',
            {success: function(response){
                Ext.Msg.alert('信息', response.responseText, function(){
                    store.reload();
                });
            },failure: function(){
                Ext.Msg.alert("错误", "与后台联系时出现了问题");
            }},
            'data=' + encodeURIComponent(Ext.encode(jsonArray))
        );
}
```

上面示例的大概流程是这样的，首先获得store中修改过的数据，然后放到JSON数组里，用Ajax提交给后台，最后根据后台返回的结果显示成功或失败信息。请注意，success是请求成

功后返回的信息，failure是请求失败后返回的信息，与业务处理成功或失败没有关系，这里很容易搞混淆。

上面的示例用到了数组对象的slice(start,[end])方法，该方法返回一个新数组，包含了源函数从start到end所指定的元素，但是不包括end元素，比如a.slice(0,3)。如果start为负，则将它作为length+start处理（此处length为数组的长度，比如a.slice(-3,4)，相当于a.slice(2,4)）。如果end为负，就将它作为length+end处理（此处length为数组的长度，比如a.slice(0,-1)）。如果省略end，那么slice方法将一直复制到源数组结尾，比如a.slice(1)。如果end出现在start之前，不复制任何元素到新数组中，比如a.slice(4,3)。示例中store.modified.slice(0)的作用就是复制store.modified，保证store.modified中的原始数据不受影响。

下面把这些数据组装成简单的数组，因为数组m里保存的都是Record，而不是简单对象，只需要取出Record的data属性即可。虽然使用普通循环也能实现，但是这里我们还是讲解一下EXT提供的each()函数。

Ext.each(array, fn)的作用是遍历array，并对每项分别调用fn函数。如果array不是数组，则只执行一次。如果某项调用fn后结果返回false（必须是false，undefined无效），那么遍历将结束并退出，后面的array项将不会被遍历。遍历过程中，每次为fn传入的参数分别为当前遍历的数组元素，当前元素索引和包含原数组所有数据的array变量。

提交数据的代码部分比较简单，我们为Ajax设置匹配的method和url，以及处理成功和失败事件的回调函数，再将EditorGrid中的数据组装成JSON形式的字符串，然后提交给后台。最后，在Ajax调用成功之后执行store.reload()，刷新表格中的数据。

另外，介绍一下store的参数pruneModifiedRecords。如果把它设置为true，每次进行remove或load操作时store会自动清除modified标记，可以避免出现下次提交时还会把上次那些modified信息都带上的现象。

完整代码如下所示：

```
var cm = new Ext.grid.ColumnModel([{
    header: '编号',
    dataIndex: 'id',
    editor: new Ext.grid.GridEditor(
        new Ext.form.TextField({
            allowBlank: false
        })
    )
}, {
    header: '名称',
    dataIndex: 'name',
    editor: new Ext.grid.GridEditor(
        new Ext.form.TextField({
            allowBlank: false
        })
    )
}, {
    header: '描述',
    dataIndex: 'descn',
```

```
        editor: new Ext.grid.GridEditor(
            new Ext.form.TextField({
                allowBlank: false
            })
        )
    }]);

    var data = [
        ['1','name1','descn1'],
        ['2','name2','descn2'],
        ['3','name3','descn3'],
        ['4','name4','descn4'],
        ['5','name5','descn5']
    ];

    var store = new Ext.data.Store({
        proxy: new Ext.data.MemoryProxy(data),
        reader: new Ext.data.ArrayReader({}, [
            {name: 'id'},
            {name: 'name'},
            {name: 'descn'}
        ]),
        pruneModifiedRecords: true
    });

    var Record = Ext.data.Record.create([
        {name: 'id', type: 'string'},
        {name: 'name', type: 'string'},
        {name: 'descn', type: 'string'}
    ]);
    store.load();

    var grid = new Ext.grid.EditorGridPanel({
        autoHeight: true,
        renderTo: 'grid',
        store: store,
        cm: cm,
        tbar: new Ext.Toolbar(['-', {
            text: '添加一行',
            handler: function(){

                var initValue = {id:'',name:'',descn:''};

                var p = new Record(initValue);

                grid.stopEditing();
                store.insert(0, p);
                grid.startEditing(0, 0);

                p.dirty = true;
                p.modified = initValue;
                if(store.modified.indexOf(p) == -1){
                    store.modified.push(p);
                }
            }
```

```
        }, '-', {
            text: '删除一行',
            handler: function(){
                Ext.Msg.confirm('信息', '确定要删除? ', function(btn){
                    if (btn == 'yes') {
                        var sm = grid.getSelectionModel();
                        var cell = sm.getSelectedCell();

                        var record = store.getAt(cell[0]);
                        store.remove(record);
                    }
                });
            }
        }, '-', {
            text: '保存',
            handler: function(){
                var m = store.modified.slice(0);
                var jsonArray = [];
                Ext.each(m, function(item) {
                    jsonArray.push(item.data);
                });

                Ext.lib.Ajax.request(
                    'POST',
                    '11_03_01.jsp',
                    {success: function(response){
                        Ext.Msg.alert('信息', response.responseText, function(){
                            store.reload();
                        });
                    },failure: function(){
                        Ext.Msg.alert("错误", "与后台联系的时候出现了问题");
                    }},
                    'data=' + encodeURIComponent(Ext.encode(jsonArray))
                );
            }
        }, '-'])
    });
```

该示例在03.grid/11-03-01.html中。

3.11.4 验证 EditGrid 中的数据

不知道大家有没有注意到，这里设置的`allowBlank:false`限制只能修改已有数据的单元格。如果新建了一行，里面的空白是无法检测到的。客户没有修改这些空白，但仍然可以直接提交数据，这样把无效数据提交到后台显然是不可取的。

很可惜的是，默认情况下，EditorGrid并没有实现这部分功能，需要我们自己在提交前进行判断，实现方法如代码清单3-7所示。

代码清单3-7　数据校验

```
for (var i = 0; i < m.length; i++) {
    var record = m[i];
```

```
        var fields = record.fields.keys;
        for (var j = 0; j < fields.length; j++) {
            var name = fields[j];
            var value = record.data[name];
            var colIndex = cm.findColumnIndex(name);
            var rowIndex = store.indexOfId(record.id);
            var editor = cm.getCellEditor(colIndex).field;

            if (!editor.validateValue(value)) {
                Ext.Msg.alert('提示', '请确保输入的数据正确。', function(){
                    grid.startEditing(rowIndex, colIndex);
                });
                return;
            }
        }
    }
```


下面将示例代码中的重点部分讲解一下。

首先，循环数组m，获得被修改的每行。var record = m[i];代表某一行，var fields =record.fields.keys;表示一共有多少列。

其次，循环列fields，获得当前行的每一个单元格。代码中获得的变量值有name（列名）、value（单元格的数值）、colIndex（列号）、rowIndex（行号），以及editor（colIndex列使用的编辑器）。

最后，进行数据校验。直接调用editor.isValid()方法总会出现el不存在的错误，所以只能使用editor.validateValue(value)的方式进行校验，这是isValid()内部的实现方法。

如果校验成功，则进入下一个单元格，重复上述步骤。如果校验失败，弹出提示对话框，并激活那个单元格的编辑状态。

经过总结发现，这里的难点是如何确定某个单元格的数据和对应的编辑器，其实store和colModel之间是相互关联的，只要认真查阅一下API，一切问题都会迎刃而解。

完整代码如下所示：

```
    var cm = new Ext.grid.ColumnModel([{
        header: '编号',
        dataIndex: 'id',
        editor: new Ext.grid.GridEditor(
            new Ext.form.TextField({
                allowBlank: false
            })
        )
    }, {
        header: '名称',
        dataIndex: 'name',
        editor: new Ext.grid.GridEditor(
            new Ext.form.TextField({
                allowBlank: false
            })
        )
```

```
}, {
    header: '描述',
    dataIndex: 'descn',
    editor: new Ext.grid.GridEditor(
        new Ext.form.TextField({
            allowBlank: false
        })
    )
}]);

var data = [
    ['1','name1','descn1'],
    ['2','name2','descn2'],
    ['3','name3','descn3'],
    ['4','name4','descn4'],
    ['5','name5','descn5']
];

var store = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id'},
        {name: 'name'},
        {name: 'descn'}
    ])
});

var Record = Ext.data.Record.create([
    {name: 'id', type: 'string'},
    {name: 'name', type: 'string'},
    {name: 'descn', type: 'string'}
]);
store.load();

var grid = new Ext.grid.EditorGridPanel({
    autoHeight: true,
    renderTo: 'grid',
    store: store,
    cm: cm,
    tbar: new Ext.Toolbar(['-', {
        text: '添加一行',
        handler: function(){

            var initValue = {id:'',name:'',descn:''};

            var p = new Record(initValue);

            grid.stopEditing();
            store.insert(0, p);
            grid.startEditing(0, 0);

            p.dirty = true;
            p.modified = initValue;
            if(store.modified.indexOf(p) == -1){
```

```
                store.modified.push(p);
            }
        }
    }, '-', {
        text: '删除一行',
        handler: function(){
            Ext.Msg.confirm('信息', '确定要删除？', function(btn){
                if (btn == 'yes') {
                    var sm = grid.getSelectionModel();
                    var cell = sm.getSelectedCell();

                    var record = store.getAt(cell[0]);
                    store.remove(record);
                }
            });
        }
    }, '-', {
        text: '保存',
        handler: function(){
            var m = store.modified.slice(0);
            for (var i = 0; i < m.length; i++) {
                var record = m[i];
                var fields = record.fields.keys;

                for (var j = 0; j < fields.length; j++) {
                    var name = fields[j];
                    var value = record.data[name];

                    var colIndex = cm.findColumnIndex(name);
                    var rowIndex = store.indexOfId(record.id);
                    var editor = cm.getCellEditor(colIndex).field;

                    if (!editor.validateValue(value)) {
                        Ext.Msg.alert('提示', '请确保输入的数据正确。', function(){
                            grid.startEditing(rowIndex, colIndex);
                        });
                        return;
                    }
                }
            }
            // 进行到这里，说明数据都是有效的
            var jsonArray = [];
            Ext.each(m, function(item) {
                jsonArray.push(item.data);
            });

            Ext.lib.Ajax.request(
                'POST',
                'grid2.jsp',
                {success: function(response){
                    Ext.Msg.alert('信息', response.responseText, function(){
                        store.reload();
```

3

```
                        });
                    },failure: function(){
                        Ext.Msg.alert("错误", "与后台联系的时候出现了问题");
                    }},
                    'data=' + encodeURIComponent(Ext.encode(jsonArray))
                );
            }
        }, '-'])
});
```

该示例在03.grid/11-04-01.html中。

3.11.5 限制输入数据的类型

任何软件或系统都会对输入的数据有所限制，比如年龄，必须是数字类型的。可以通过输入方式直接对输入的数据类型进行限制，从而保证客户不会输入完全错误的数据，而且这样做比较简单直观。

EXT提供了多种数据类型的组件，比如NumberField限制只能输入数字、ComboBox限制只能输入备选项、DateField是选择日期、Checkbox则是从true和false中选择其一，等等。

我们现在来修改之前的数据模型，让数据类型变得更丰富，如下面的代码所示。

```
// 放到grid里显示的原始数据
var data = [
    [1.1,1,new Date(),true],
    [2.2,2,new Date(),false],
    [3.3,0,new Date(),true],
    [4.4,1,new Date(),false],
    [5.5,2,new Date(),true]
];
```

第1列是数字，第2列是ComboBox对应的id，第3列是日期，第4列是布尔型。

现在分别对这5列设置各自的编辑器，第1列限制只能用数字，不能是负数，而且不能超过10，如下面的代码所示。

```
var cm = new Ext.grid.ColumnModel([{
    header:'数字列',
    dataIndex:'number',
    editor:new Ext.grid.GridEditor(new Ext.form.NumberField({
        allowBlank: false,
        allowNegative: false,
        maxValue: 10
    }))
},
......
```

使用NumberField就表示只能输入数字，allowBlank:false表示不能为空，allowNegative:false表示不能输入减号，maxValue:10表示可输入的最大值，运行结果如图3-32所示。

数字列	选择列	日期列	判断列
1.1	ext在线支持	2008-04-25	是
2.2	ext扩展	2008-04-25	否
3.3	新版ext教程	2008-04-25	是
4.4	ext在线支持	2008-04-25	否
5.5	ext扩展	2008-04-25	是

图3-32 限制输入数据类型

第2列稍微复杂一些，由于是ComboBox，所以我们得先准备下拉列表里的数据，如下所示。

```
var comboData = [
    ['0','新版ext教程'],
    ['1','ext在线支持'],
    ['2','ext扩展']
];
```

数据部分单独抽离出来是为了在editor和renderer里保持数据同步，配置的参数都已经在前面与ComboBox相关的内容中介绍过。

ComboBox的配置如代码清单3-8所示。

代码清单3-8 配置ComboBox

```
{
    header:'选择列',
    dataIndex:'combo',
    editor:new Ext.grid.GridEditor(
new Ext.form.ComboBox({
        store: new Ext.data.SimpleStore({
            fields:['value','text'],
            data: comboData
        }),
        emptyText: '请选择',
        mode: 'local',
        triggerAction: 'all',
        valueField: 'value',
        displayField: 'text',
        readOnly:true
    })),
    renderer: function(value){
        return comboData[value][1];
    }
}
```

大家仔细研究一下渲染函数renderer，经常有人会遇到EditorGrid里的ComboBox总是无法正常显示数据的情况。其实，就是因为少了这个renderer函数。当没写这个函数时，显示的数据是value值，而不是text。毕竟EditorGrid里的编辑器只在实际编辑时起作用，表格与editor之间共享的是数据，显示层都要依靠各自的实现。不过这样做更灵活，不需要两者都使用一样的显示方式，如图3-33所示。

数字列	选择列	日期列	判断列
1.1	ext在线支持	2008-04-25	是
2.2	新版ext教程	2008-04-25	否
3.3	ext在线支持	2008-04-25	是
4.4	ext扩展	2008-04-25	否
5.5	ext扩展	2008-04-25	是

图3-33　ComboBox

第3列是选择日期，这也将是大家常用到的控件，具体代码如下所示。

```
{
    header:'日期列',
    dataIndex:'date',
    editor:new Ext.grid.GridEditor(new Ext.form.DateField({
        format: 'Y-m-d',
        minValue: '2007-12-14',
        disabledDays: [0, 6],
        disabledDaysText: '只能选择工作日'
    })),
    renderer: function(value) {
        return value.format("Y-m-d");
    }
}
```

比较常用的有format:'Y-m-d'，这样显示的日期会变成“年-月-日”的格式；minValue设置日期最小值，disabledDays是很有趣的一个选项，你可以通过它禁止用户选择星期几，我们这里禁用了周六和周日。同时也改一下提示信息：“只能选择工作日”。如图3-34所示。

数字列	选择列	日期列	判断列
1.1	ext在线支持	2008-04-25	是
2.2	ext扩展		
3.3	新版ext教程		
4.4	ext在线支持		
5.5	ext扩展		

图3-34　DateField

第4列中的checkbox就只能选择true或false，所以也没有什么需要配置的，如下面的代码所示。

```
{
    header:'判断列',
    dataIndex:'check',
    editor:new Ext.grid.GridEditor(new Ext.form.Checkbox({
        allowBlank: false
```

```
    })),
    renderer:function(value) {
        return value ? '是' : '否';
    }
}]);
```

尽管不是必须的，我们还是用renderer对显示进行了美化，如图3-35所示。

数字列	选择列	日期列	判断列
1.1	ext在线支持	2008-04-25	☑
2.2	ext扩展	2008-04-25	否
3.3	新版ext教程	2008-04-25	是
4.4	ext在线支持	2008-04-25	否
5.5	ext扩展	2008-04-25	是

图3-35 CheckBox

完整代码如下所示：

```
var comboData = [
    ['0','新版ext教程'],
    ['1','ext在线支持'],
    ['2','ext扩展']
];

var cm = new Ext.grid.ColumnModel([{
    header:'数字列',
    dataIndex:'number',
    editor:new Ext.grid.GridEditor(new Ext.form.NumberField({
        allowBlank: false,
        allowNegative: false,
        maxValue: 10
    }))
},{
    header:'选择列',
    dataIndex:'combo',
    editor:new Ext.grid.GridEditor(new Ext.form.ComboBox({
        store: new Ext.data.SimpleStore({
            fields:['value','text'],
            data: comboData
        }),
        emptyText: '请选择',
        mode: 'local',
        triggerAction: 'all',
        valueField: 'value',
        displayField: 'text',
        readOnly:true
    })),
    renderer: function(value){
        return comboData[value][1];
    }
},{
    header:'日期列',
```

```
        dataIndex:'date',
        editor:new Ext.grid.GridEditor(new Ext.form.DateField({
            format: 'Y-m-d',
            minValue: '2007-12-14',
            disabledDays: [0, 6],
            disabledDaysText: '只能选择工作日'
        })),
        renderer: function(value) {
            return value.format("Y-m-d");
        }
    },{
        header:'判断列',
        dataIndex:'check',
        editor:new Ext.grid.GridEditor(new Ext.form.Checkbox({
            allowBlank: false
        })),
        renderer:function(value) {
            return value ? '是' : '否';
        }
    }]);

    // 放到grid里显示的原始数据
    var data = [
        [1.1,1,new Date(),true],
        [2.2,2,new Date(),false],
        [3.3,0,new Date(),true],
        [4.4,1,new Date(),false],
        [5.5,2,new Date(),true]
    ];

    var store = new Ext.data.Store({
        proxy: new Ext.data.MemoryProxy(data),
        reader: new Ext.data.ArrayReader({}, [
            {name: 'number'},
            {name: 'combo'},
            {name: 'date'},
            {name: 'check'}
        ])
    });
    store.load();

    var grid = new Ext.grid.EditorGridPanel({
        autoHeight: true,
        renderTo: 'grid',
        store: store,
        cm: cm
    });
```

该示例在03.grid/11-05-01.html中。

3.12 属性表格控件——PropertyGrid

PropertyGrid是在EditorGrid的基础上开发的更智能易用的高级表格组件。下面将介绍

PropertyGrid的功能和应用方式。

3.12.1 PropertyGrid

先来看看图3-36。

属性表格

Name ▲	Value
创建时间	12/15/2007
名字	不告诉你
描述	嗯，估计没啥可说的
是否有效	true
版本号	0.01

图3-36 属性表格

Property Grid（属性表格）扩展自EditorGridPanel，所以可以直接编辑右边的内容。注意，只有右边的，即使你单击左边的单元格，编辑器也只会出现在右边。

实际上，我们可以用散列表来形容PropertyGrid，左边可以看作key，右边的是value。key是由我们指定好的，用户只需要修改对应的value即可。对某些属性而言，配置很方便。

代码方面，因为只有两列，所以我们不用担心columnModel，ds的部分确定是key和value的组合，因此也不用再分几步准备。下面我们来创建一个PropertyGrid，如代码清单3-9所示。

代码清单3-9 创建PropertyGrid

```
var grid = new Ext.grid.PropertyGrid({
    title: '属性表格',
    autoHeight: true,
    width: 300,
    renderTo: 'grid',
    source: {
        "名字": "不告诉你",
        "创建时间": new Date(Date.parse('12/15/2007')),
        "是否有效": false,
        "版本号": .01,
        "描述": "嗯，估计没啥可说的"
    }
});
```

title就是整个表格的标题；autoHeight很有用，这样就不需要我们为div指定style；width是需要的宽度；renderTo应该很眼熟，它指定渲染的元素id。剩下的就是source，这个JSON数据里指定了一组key和value，最后它们都会显示到表格里。

PropertyGrid提供的功能还不只这些，试着单击一下单元格，你会发现string、date、bool和number对应着默认的编辑器，分别是TextField、DateField、ComboBox和NumberField，bool对应的ComboBox里只有true和false两项。

该示例在03.grid/12-01-01.html中。

3.12.2 只能看不能动的PropertyGrid

PropertyGrid默认提供编辑功能，但是在某些情况下，只用它来显示数据，这时就需要禁用PropertyGrid中的编辑功能。PropertyGrid中并没有直接提供可以开关编辑功能的参数，但我们可以通过EXT的事件监听器实现这一功能，代码如下。

```
grid.on("beforeedit", function(e){
    e.cancel = true;
    return false;
});
```

该示例在03.grid/12-02-01.html中。

3.12.3 强制对 name 列排序

看到name列上的排序箭头，PropertyGrid对source中的第一列自动使用升序排列，这就造成了最终的显示顺序很可能与我们输入的顺序不同。

有很多人以为在EXT健壮的控件库内部肯定有一个选项用来控制排序，其实它的排序方法是写死在代码里的，而且没有参数可以控制。如果想修改排序方式，就只能修改源码，如代码清单3-10所示。

代码清单3-10 修改默认排序

```
Ext.grid.PropertyGrid.prototype.initComponent = function(){
    this.customEditors = this.customEditors || {};
    this.lastEditRow = null;
    var store = new Ext.grid.PropertyStore(this);
    this.propStore = store;
    var cm = new Ext.grid.PropertyColumnModel(this, store);
    // store.store.sort('name', 'ASC');
    this.addEvents(
        'beforepropertychange',
        'propertychange'
    );
    this.cm = cm;
    this.ds = store.store;
    Ext.grid.PropertyGrid.superclass.initComponent.call(this);

    this.selModel.on('beforecellselect', function(sm, rowIndex, colIndex){
        if(colIndex === 0){
            this.startEditing.defer(200, this, [rowIndex, 1]);
            return false;
        }
    }, this);
};
```

写了这么多代码，结果只是为了注释掉其中的store.store.sort('name', 'ASC');，让它不再对name进行升序排序。

该示例在03.grid/12-03-01.html中。

3.12.4 根据 name 获得 value

PropertyGrid既然已经是一个散列表的形式，自然会想到直接通过name获得value。可惜PropertyGrid没有直接提供这个功能，需要我们自己去store里找，如下面的代码所示。

```
grid.store.getById('名字').get('value');
```

先获得id为“名字”的那一行，然后获得value列的值。如果能提供一个快捷的函数，那就更好。

该示例在03.grid/12-04-01.html中。

3.12.5 自定义编辑器

默认的TextField、DateField、ComboBox和NumberField也就只能处理一般的情况。当我们想对编辑器进行更详细的配置时，就需要用到PropertyGrid的customEditors，为指定id的那行数据设置对应的编辑器。让我们按照API里的示例来制作一个TimeField吧。

```
var grid = new Ext.grid.PropertyGrid({
    title: '属性表格',
    autoHeight: true,
    width: 300,
    renderTo: 'grid',
    customEditors: {
        'Start Time':new Ext.grid.GridEditor(new
Ext.form.TimeField({selectOnFocus:true}))
    },
    source: {
        'Start Time': '10:00 AM'
    }
});
```

customEditors和source的设置基本一样，只需要将两者的属性名称对应起来，并且为customEditors里的所有属性指定一个editor。这样我们就获得了自制的编辑器，与默认编辑器的功能一样，如图3-37所示。

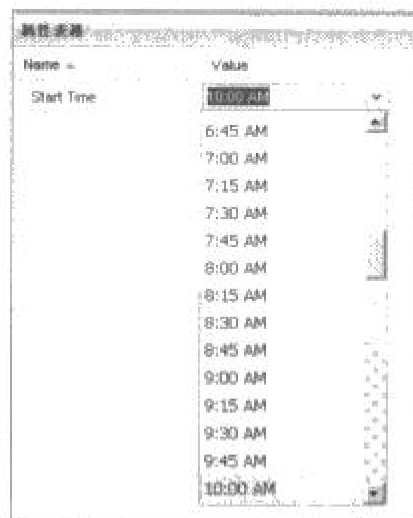


图3-37 TimeField

这样我们就获得对编辑器的完全控制权限，可以使用我们自己需要的组件。

该示例在03.grid/12-05-01.html中。

3.13 分组表格控件——Group

Ext.grid.GroupingGrid分组表格就是在普通表格的基础上，根据某一列的数据将表格中的数据分组显示的表格控件。

3.13.1 分组表格简介

像图3-38这样带有分组功能的表格还是很有用处的。

编号 ▲	性别	名称	描述
⊟ 性别: female			
2	female	name2	descn2
4	female	name4	descn4
⊟ 性别: male			
1	male	name1	descn1
3	male	name3	descn3
5	male	name5	descn5

图3-38 分组表格

首先定义一组数据，如下面的代码所示：

```
var data = [
    ['1','male','name1','descn1'],
    ['2','female','name2','descn2'],
    ['3','male','name3','descn3'],
    ['4','female','name4','descn4'],
    ['5','male','name5','descn5']
];
```

这些数据将根据第2列“性别”来进行分组，然后还要根据id进行升序排列，通过Ext.data.GroupingStore实现分组效果，最后提供给GridPanel，如下面代码所示。

```
var store = new Ext.data.GroupingStore({
    reader: reader,
    data: data,
    groupField: 'sex',
    sortInfo: {field: 'id', direction: "ASC"}
});
```

这里的reader和data还是和的原来的示例一样，需要关注的是groupField，它确定通过哪一项分组。令人困惑的是，GroupingStore要求必须设置sortInfo。这个参数对应的值里有两项，field是排序的属性，direction是排序的方式。我们把数据传入，输出显示的就是分成一组一组的数据。但是，如果想显示成我们期望的那种形式，还需要设置GroupingView，如下面的代码所示。

```
var grid = new Ext.grid.GridPanel({
    store: store,
    columns: columns,
    view: new Ext.grid.GroupingView(),
    renderTo: 'grid'
});
```

该示例在03.grid/13-01.html中。

此外，还有GroupingSummary.js和RowExpand.js，不过这些不在ext-all.js中，用到时需要另外导入。

3.13.2 分组表格视图 Ext.grid.GroupingView

Ext.grid.GroupingView继承自Ext.grid.GridView，它是分组表格的显示视图，实际应用中需要与Ext.data.GroupingStore配合使用。为了在Ext.grid.GridPanel中使用Ext.grid.GroupingView，我们要在创建Ext.grid.GridPanel时传入一个Ext.grid.GroupingView的实例，如下面的代码所示。

```
var grid = new Ext.grid.GridPanel({
    store: new Ext.data.GroupingStore({
        reader: new Ext.data.ArrayReader({}, meta),
        data: data,
        groupField: 'sex',
        sortInfo: {field: 'id', direction: "ASC"}
    }),
    columns: meta,
    view: new Ext.grid.GroupingView(),
    renderTo: 'grid',
    autoHeight: true
});
```

其中view: new Ext.grid.GroupingView()是在Ext.grid.GridPanel中使用分组视图的必要条件。在之后的代码中我们会使用grid.getView()获得Ext.grid.GroupingView的实例，可以在它的实例上调用功能函数，对表格的视图部分进行操作。比较常用的操作是展开或折叠分组，如下面的代码所示。

```
Ext.get('expand').on('click', function() {
    grid.getView().expandAllGroups();
});
Ext.get('collapse').on('click', function() {
    grid.getView().collapseAllGroups();
});
Ext.get('toggle').on('click', function() {
    grid.getView().toggleAllGroups();
});
```

我们在3个按钮上添加了监听事件，在单击不同按钮时对分组视图进行展开或折叠操作。上例中，expandAllGroups()展开所有分组，collapseAllGroups()折叠所有分组，toggleAllGroups()会自动判断分组的状态，当分组都已折叠时会展开所有分组，当分组都已展开时会折叠所有分组。

也可以对某一分组执行展开或折叠操作，如下面的代码所示。

```
Ext.get('one').on('click', function() {
    var view = grid.getView();
    var groupId = view.getGroupId('female');
    view.toggleGroup(groupId);
});
```

上例中，首先获得Ext.grid.GroupingView的实例，然后调用getGroupId()函数获得对应分组的id。因为之前是根据性别进行分组的，所以getGroupId('female')得到的就是女性对应的分组，在获得groupId之后就可以调用toggleGroup()函数对这个分组执行展开或折叠操作了，结果如图3-39所示。

expand collapse toggle toggle one

编号	性别	名称	描述
⊞ 性别: female			
⊟ 性别: male			
1	male	name1	descn1
3	male	name3	descn3
5	male	name5	descn5

图3-39 通过Ext.grid.GroupingView实现分组的展开和折叠功能

完整代码如下所示：

```
Ext.onReady(function(){

    Ext.QuickTips.init();

    var meta = [
        {header:'编号',dataIndex:'id', name:'id'},
        {header:'性别',dataIndex:'sex', name:'sex'},
        {header:'名称',dataIndex:'name', name:'name'},
        {header:'描述',dataIndex:'descn', name:'descn'}
    ];

    var data = [
        ['1','male','name1','descn1'],
        ['2','female','name2','descn2'],
        ['3','male','name3','descn3'],
        ['4','female','name4','descn4'],
        ['5','male','name5','descn5']
    ];

    var grid = new Ext.grid.GridPanel({
        store: new Ext.data.GroupingStore({
            reader: new Ext.data.ArrayReader({}, meta),
            data: data,
            groupField: 'sex',
            sortInfo: {field: 'id', direction: "ASC"}
        }),
        columns: meta,
        view: new Ext.grid.GroupingView(),
        renderTo: 'grid',
        autoHeight: true
    });

    Ext.get('expand').on('click', function() {
```

```
        grid.getView().expandAllGroups();
    });
    Ext.get('collapse').on('click', function() {
        grid.getView().collapseAllGroups();
    });
    Ext.get('toggle').on('click', function() {
        grid.getView().toggleAllGroups();
    });
    Ext.get('one').on('click', function() {
        var view = grid.getView();
        var groupId = view.getGroupId('female');
        view.toggleGroup(groupId);
    });
});
```

该示例在03.grid/13-02.html中。

3.14 可拖放的表格

本节介绍表格中的拖曳功能，包括拖曳改变表格大小。拖动表格中的某一行进行排序，将某一行由一个表格中拖到另一个表格中，最后还演示了如何在表格和树形之间进行拖曳。

3.14.1 拖放改变表格的大小

注意图3-40下面蓝色的细条，把鼠标放到上面，就可以任意改变表格的高度。实现这种效果并不难，也不需要对写好的表格做大的修改，如下面代码所示。

```
var rz = new Ext.Resizable('grid', {
    wrap:true,
    minHeight:100,
    pinned:true,
    handles: 's'
});
rz.on('resize', grid.syncSize, grid);
```

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

图3-40 拖放改变表格大小

Resizable必须放在render之后，否则就会出现问题。下面我们来看看参数的构成。

- 第一个参数是'grid'，就是说这个可改变大小的区域是在div id="grid"这个元素上起作用。
- wrap:true，这个参数会在构造Resizable时自动在指定id的外边包裹一层div，这样就

不用在HTML里定义其他附属的div了。

- minHeight:100，它限制改变的最小高度。
- pinned:true，此参数控制可拖动区域的显示状态。如果值为true，则可拖动区域会一直显示在表格下方；如果为false，只有鼠标悬停在可拖曳区域上方时才会出现。具体配置取决于个人喜好。
- handlers: 's'，'s'即'south'。EXT中用东、南、西、北对应上、下、左、右，用首字母来设置可以拖放的方向，具体的配置值如表3-1所示。
- 最后别忘了注册事件rz.on('resize', grid.syncSize, grid);，在拖放完成之后，表格会调用syncSize方法修改自己的大小，第三个参数是函数执行的scope。

表3-1　Resizable的handlers配置

值	英　文	中　文	值	英　文	中　文
'n'	north	北	'sw'	southwest	西南
's'	south	南	'se'	southeast	东南
'e'	east	东	'ne'	northeast	东北
'w'	west	西	'all'	all	所有方向
'nw'	northwest	西北			

完整代码如下所示：

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
]);

var data = [
    ['1','name1','descn1'],
    ['2','name2','descn2'],
    ['3','name3','descn3'],
    ['4','name4','descn4'],
    ['5','name5','descn5']
];

var store = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id'},
        {name: 'name'},
        {name: 'descn'}
    ])
});
store.load();

var grid = new Ext.grid.GridPanel({
    renderTo: 'grid',
```

```
        store: store,
        cm: cm
    });

    var rz = new Ext.Resizable(grid.getEl(), {
        wrap: true,
        minHeight:100,
        pinned:true,
        handles: 's'
    });
    rz.on('resize', grid.syncSize, grid);
```

该示例在03.grid/14-01-01.html中。

3.14.2 在同一个表格里拖放

在同一个表格里拖放只能用在排序上，而且还必须是在少量数据的情况下。不过，因为EXT的表格内置了对拖放的支持，所以使用起来非常方便，只需要把enableDragDrop设置为true就行了。

如下面代码所示：

```
var grid = new Ext.grid.GridPanel({
    renderTo: 'grid',
    store: store,
    cm: cm,
    enableDragDrop: true
});
```

如此一来，你就可以任意拖动了。不过你会发现，拖起来的行不能放下，总会显示一个禁止放下的图标，如图3-41所示。

编号	名称	描述
1	name1	descn1
4	name4	descn4
5	name5	descn5
2	name2	descn2
3	name3	descn3

3 selected rows

图3-41 拖放行

虽然表格内置了拖放开关，却没有默认的拖放处理函数，我们必须添加自定义的处理函数，才能让拖起来的行能放下去。

拖起来的行放不下去是因为表格里没有设置DropTarget，这就是放置被拖动数据的目标。比如一堆苹果和一个篮子，要把苹果放到篮子里。那么DropTarget相当于篮子，DragTarget相当于苹果，把苹果（DragTarget）拿起来，然后放到篮子（DropTarget）里去。这样讲大家应该比较清楚了吧。

现在设置一个DropTarget，为了实现在同一表格中进行拖曳，我们就把当前表格作为DropTarget。

对grid.container进行如下设置（如代码清单3-11所示），让表格的容器成为DropTarget。

代码清单3-11 设置DropTarget

```
var ddrow = new Ext.dd.DropTarget(grid.container, {
    ddGroup : 'GridDD',
```

```
        copy     : false,
        notifyDrop : function(dd, e, data) {
            // 选中了多少行
            var rows = data.selections;
            // 拖动到第几行
            var index = dd.getDragData(e).rowIndex;
            if (typeof(index) == "undefined") {
                return;
            }
            // 修改store
            for(i = 0; i < rows.length; i++) {
                var rowData = rows[i];
                if(!this.copy) store.remove(rowData);
                store.insert(index, rowData);
            }
        }
});
```

这样就把grid.container变成了DropTarget。看到ddGroup了吗？这个分组名称必须与DragZone里的分组一样，这样才能有效果。表格里默认的ddGroup就叫做'GridDD'，把这两个写成一样即可。

copy:false决定了拖放以后是执行剪切操作还是复制操作。如果是copy:false，拖过去以后，原来的数据就应该不见了。如果是copy:true，原来的数据不会发生变化，这里当然要选择copy:false。

然后，notifyDrop对应的函数将处理拖放事件，这个函数主要分成如下3部分。

- 获得你选中的那几行。
- 根据拖放事件获得拖放目标的行索引，不过得到的索引可能是undefined，此时我们就不处理它。
- 把刚刚选中的那些数据拖放到想放的位置，如果是copy:false，先删除再添加数据，否则就不会删除数据。

完整代码如下所示：

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
]);

var data = [
    ['1','name1','descn1'],
    ['2','name2','descn2'],
    ['3','name3','descn3'],
    ['4','name4','descn4'],
    ['5','name5','descn5']
];

var store = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, [
```

```
            {name: 'id'},
            {name: 'name'},
            {name: 'descn'}
        ])
    });
    store.load();

    var grid = new Ext.grid.GridPanel({
        autoHeight: true,
        renderTo: 'grid',
        store: store,
        cm: cm,
        enableDragDrop: true
    });

    var ddrow = new Ext.dd.DropTarget(grid.container, {
        ddGroup : 'GridDD',
        copy    : false,
        notifyDrop : function(dd, e, data) {
            // 选中了多少行
            var rows = data.selections;
            // 拖动到第几行
            var index = dd.getDragData(e).rowIndex;
            if (typeof(index) == "undefined") {
                return;
            }
            // 修改store
            for(i = 0; i < rows.length; i++) {
                var rowData = rows[i];
                if(!this.copy) store.remove(rowData);
                store.insert(index, rowData);
            }
        }
    });
```

该示例在03.grid/14-02-01.html中。

3.14.3 表格之间的拖放

表格里的数据还可以跨表格拖动，也就是表格之间可以互相拖动。下面还会讲到表格与树之间的拖动。

首先，创建两个表格，当然都要设置enableDragDrop:true。再在这个基础上配置DropTarget，然后就可以任意拖动了，如下面的代码所示。

```
    var grid1 = new Ext.grid.GridPanel({
        renderTo: 'grid1',
        store: store1,
        cm: cm,
        enableDragDrop: true
    });
    var grid2 = new Ext.grid.GridPanel({
        renderTo: 'grid2',
        store: store2,
```

```
        cm: cm,
        enableDragDrop: true
    });
```

这里创建两个表格grid1和grid2，分别对应着各自的Ext.data.Store。然后，分别为它们创建各自的DropTarget。这里的修改仅针对最后的转移数据。下面是对应grid1的DropTarget，它会从store2中删除数据，然后把这些数据添加到store1里，如下面代码所示。

```
// 修改store
for(i = 0; i < rows.length; i++) {
    var rowData = rows[i];
    if(!this.copy) store2.remove(rowData);
    store1.insert(index, rowData);
}
```

完整代码如下所示：

```
var store1 = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy([
        ['01','name01','descn01'],
        ['02','name02','descn02'],
        ['03','name03','descn03'],
        ['04','name04','descn04'],
        ['05','name05','descn05']
    ]),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id'},
        {name: 'name'},
        {name: 'descn'}
    ])
});
var store2 = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy([
        ['11','name11','descn11'],
        ['12','name12','descn12'],
        ['13','name13','descn13'],
        ['14','name14','descn14'],
        ['15','name15','descn15']
    ]),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id'},
        {name: 'name'},
        {name: 'descn'}
    ])
});

store1.load();
store2.load();

var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
]);
```

```
var grid1 = new Ext.grid.GridPanel({
    autoHeight: true,
    renderTo: 'grid1',
    store: store1,
    cm: cm,
    enableDragDrop: true
});
var grid2 = new Ext.grid.GridPanel({
    autoHeight: true,
    renderTo: 'grid2',
    store: store2,
    cm: cm,
    enableDragDrop: true
});

var ddrow1 = new Ext.dd.DropTarget(grid1.view.mainBody, {
    ddGroup : 'GridDD',
    copy    : false,
    notifyDrop : function(dd, e, data) {
        // 选中了多少行
        var rows = data.selections;
        // 拖动到第几行
        var index = dd.getDragData(e).rowIndex;
        if (typeof(index) == "undefined") {
            index = 0;
        }
        // 修改store
        for(i = 0; i < rows.length; i++) {
            var rowData = rows[i];
            if(!this.copy) store2.remove(rowData);
            store1.insert(index, rowData);
        }
    }
});

var ddrow2 = new Ext.dd.DropTarget(grid2.view.mainBody, {
    ddGroup : 'GridDD',
    copy    : false,
    notifyDrop : function(dd, e, data) {
        // 选中了多少行
        var rows = data.selections;
        // 拖动到第几行
        var index = dd.getDragData(e).rowIndex;
        if (typeof(index) == "undefined") {
            index = 0;
        }
        // 修改store
        for(i = 0; i < rows.length; i++) {
            var rowData = rows[i];
            if(!this.copy) store1.remove(rowData);
            store2.insert(index, rowData);
        }
    }
});
```

该示例在03.grid/14-03-01.html中。

3.14.4　表格与树之间的拖放[1]

在实现这个功能时，下面这些步骤需要特别注意。

(1) 首先创建一个表格，然后创建一个树形，这样才能实现它们之间的拖放。

(2) 在表格里加上enableDragDrop:true，让表格支持拖放。使用grid.container构造DropTarget，让树节点可以放置在表格上。

(3) 在树形中设置enableDD:true，从而启用拖放功能。对应的事件是nodedragover和beforenodedrop。

(4) 设置ddGroup，如我们之前所说的，只有让表格和树形处于同一ddGroup中才能成功拖放。表格默认的ddGroup是GridDD，树形默认的ddGroup是TreeDD，只需要修改其中之一即可。我们在这里把树形的ddGroup改成GridDD。

(5) 在设置了这些之后，要确保grid.render()在tree.render()之后才能让拖放生效，否则树节点不能拖曳到表格上。

完整代码如下所示：

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},
    {header:'名称',dataIndex:'name'},
    {header:'描述',dataIndex:'descn'}
]);

var data = [
    ['1','name1','descn1'],
    ['2','name2','descn2'],
    ['3','name3','descn3'],
    ['4','name4','descn4'],
    ['5','name5','descn5']
];

var store = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'id'},
        {name: 'name'},
        {name: 'descn'}
    ])
});
store.load();

var grid = new Ext.grid.GridPanel({
    autoHeight: true,
    renderTo: 'grid',
    store: store,
    cm: cm,
```

[1] 建议先阅读本书第5章中与“树”有关的内容后再来阅读本小节。

```
        enableDragDrop: true
    });

    var ddrow = new Ext.dd.DropTarget(grid.view.mainBody, {
        ddGroup : 'GridDD',
        copy    : false,
        notifyDrop : function(dd, e, data){
        }
    });

    var tree = new Ext.tree.TreePanel({
        el: 'tree',
        ddGroup: 'GridDD',
        enableDD: true,
        loader: new Ext.tree.TreeLoader({dataUrl: '03-04.txt'})
    });

    var root = new Ext.tree.AsyncTreeNode({
        text: '偶是根'
    });
    tree.setRootNode(root);
    tree.render();
    root.expand();

    tree.on('nodedragover', function(e) {
    });
    tree.on('beforenodedrop', function(e){
    });
```

该示例在03.grid/14-04-01.html中。

示例中没有实现拖过去放下的效果，因为树形的数据结构与表格的不同，同时还要考虑拖放父节点的情况，从表格拖放多行到树形。这些就要根据需求具体分析了。

3.15 表格与右键菜单

表格提供了4个与右键菜单有关的事件，如下所示。

- `contextmenu : ( Ext.EventObject e )`
  全局性的右键事件。
- `cellcontextmenu : ( Grid this, Number rowIndex, Number cellIndex, Ext.EventObject e )`
  单元格上的右键事件。
- `rowcontextmenu : ( Grid this, Number rowIndex, Ext.EventObject e )`
  行上的右键事件。
- `headercontextmenu : ( Grid this, Number columnIndex, Ext.EventObject e )`
  表头的右键事件。

之所以会有4个事件，是因为表格结构复杂，普通的右键事件不能精确定位右键所在的位置。如果让用户自己确定鼠标的位置，然后再去判断选择的范围，几乎是不可能完成的任务。EXT细

分了各种事件让我们可以更容易实现需求。

下面我们介绍其中一种事件rowconextmenu的使用方法，先看看图3-42吧。

编号	名称	描述
1	name1	descn1
2	name2	
3	name3	
4	name4	descn4
5	name5	descn5

查看详情

图3-42　右键菜单

看起来并没有什么特别的，我们仅仅制作了一个菜单，还需要使用rowcontextmenu把右键菜单挂在表格上。实现过程如代码清单3-12所示。

代码清单3-12　实现右键菜单

```
var contextmenu = new Ext.menu.Menu({
    id: 'theContextMenu',
    items: [{
        text: '查看详情',
        handler: function(){
        }
    }]
});
grid.on("rowcontextmenu", function(grid, rowIndex, e){
    e.preventDefault();
    grid.getSelectionModel().selectRow(rowIndex);
    contextmenu.showAt(e.getXY());
});
```

contextmenu部分只是新建了EXT提供的组件menu，并没有什么奇特的。在监听rowcontextmenu时，回调函数可以使用3个参数：grid是当前操作的表格，rowIndex表示当前鼠标所在的行号，e则是封装好的事件对象。

在触发rowcontextmenu时，首先使用e.preventDefault()防止浏览器弹出默认的右键菜单，然后调用grid.getSelectionModel().selectRow(rowIndex);选中这一行。这是我们使用rowcontextmenu的主要原因，右键单击不会触发click事件，不会在表格上选中一行，需要我们手工选择，而rowIndex正是选中行的索引。最后，contextmenu.showAt(e.getXY())就能显示菜单了。

该示例在03.grid/15-01.html中。

如果我们想得到一个单元格，就需要使用cellcontextmenu事件，它有4个参数，其中的rowIndex和cellIndex可以决定我们选中哪一个单元格。以此类推，如果需要选择表头，就应该使用headercontextmenu。如果只想为表格提供一个总的功能菜单，就可以直接选择contextmenu。

3.16 小结

本章主要介绍了如何使用EXT中的表格控件。从一个最简单的表格实例开始，一步步介绍了表格需要的各种配置。同时还讲解了表格的选择模型，通过配置不同的sm修改表格的内部行为。

对表格进行分页和排序是一种常见的功能，我们介绍了如何使用表格在前台和后台进行分页和排序，并给出了多个示例。

可编辑表格EditorGrid是另一个常见的表格控件。我们从一个最简单的EditorGrid开始，逐步讨论了如何添加和删除行，如何将修改后的数据提交到后台保存，以及如何对输入的数据进行校验和类型限制。

随后介绍了EXT特殊表格控件：PropertyGrid和GroupingGrid，并讨论了与表格相关的几种拖放形式。

最后讨论了表格上与右键菜单相关的问题，并根据示例进行了分析。

EXT 3.0中为表格提供了更多效果非凡的扩展组件，可以参考第14章了解这些扩展组件的用法。

第 4 章

表单与输入控件

4

本章内容

- 制作表单
- FormPanel和BasicForm详解
- EXT支持的控件
- 使用表单提交数据
- 数据校验
- 表单布局
- ComboBox详解
- 复选框和单选框
- 文件上传
- 自动把数据填充到表单中

表单是EXT中一道绚丽的风景线，初看那些输入控件，好像只是修改了CSS样式表而已。打开Firebug查看一下DOM，似乎确实只有CSS发生了变化，隐藏在漂亮界面下的依然是原来的Input控件。从这点来看，似乎没有使用EXT的必要。但是，你不想用默认的数据校验吗？你不想在数据校验失败时有突出的提示效果吗？你不想要超炫的下拉列表Combox吗？你不想要一些梦寐以求的选择控件吗？正因为EXT提供了这些绚丽的功能，所以我们建议你还是尝试一下EXT的表单（Form）和对应的输入控件。

4.1 制作表单

我们先来看一个制作表单的简单示例，实现过程如代码清单4-1所示。

代码清单4-1 制作一个简单的表单

```
var form = new Ext.form.FormPanel({
    defaultType: 'textfield',
    labelAlign: 'right',
    title: 'form',
    labelWidth: 50,
    buttonAlign: 'center',
    frame: true,
    width: 220,
```

```
        items: [{
            fieldLabel: '文本框'
        }],
        buttons: [{
            text: '按钮'
        }]
    });
    form.render("form");
```

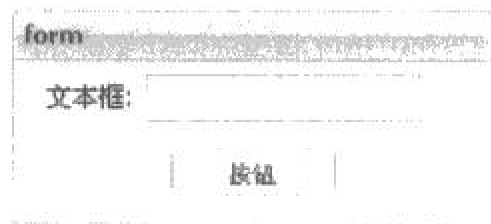


图4-1 一个简单的表单

结果如图4-1所示。

HTML里不需要那么多东西了，只需要定义一个div id="form"就可以实现这一切。初始配置显然变得更紧凑，利用items和buttons指定包含的控件和按钮。

我们先对此有一个初步的认识，稍后会有更详细的介绍，示例见04.form/01-01.html。

4.2 FormPanel 和 BasicForm 详解

如上面的示例所示，我们要制作一个Ext.form.FormPanel，然后在里面设置field。顾名思义，FormPanel是Ext.Panel的一个子类，可以对其执行各种Panel的操作。实际上，表单的功能是在Ext.form.BasicForm中实现的，在获得Ext.form.FormPanel之后，随时都可以用getForm()方法获得BasicForm对象。我们可以在得到的BasicForm上执行“提交表单数据”和“复位表单初始值”等操作。

引入Ext.form.FormPanel的最大好处就是利于布局，Ext.form.FormPanel继承了Ext.Panel。我们可以把Ext.form.FormPanel放到Ext.Viewport中作为整个页面布局的一部分，同时也可以利用items指定Ext.form.FormPanel内部的子组件。如其他Panel一样，可以通过xtype来指定每个子组件的类型，而不必手工创建它们。

4.3 EXT 支持的控件

下面我们来逐一介绍EXT中提供的各种输入控件。

4.3.1 控件继承图

EXT中提供的输入控件包括TextField、TextArea、CheckBox、Radio、ComboBox、DateField、HtmlEditor、Hidden和TimeField。所有这些控件的继承关系如图4-2所示。

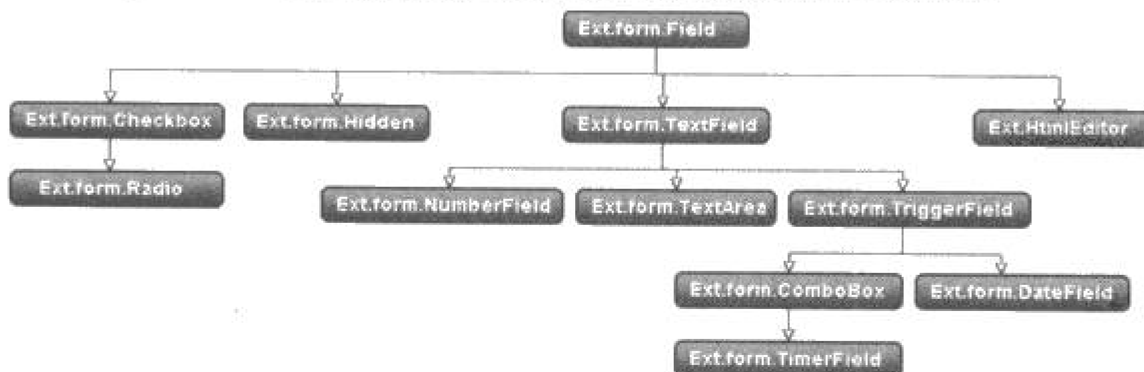


图4-2 EXT支持的控件的继承关系图

4.3.2 表单控件

图4-3展示了EXT中支持的所有组件，这些组件的用法将在后面详细讨论。

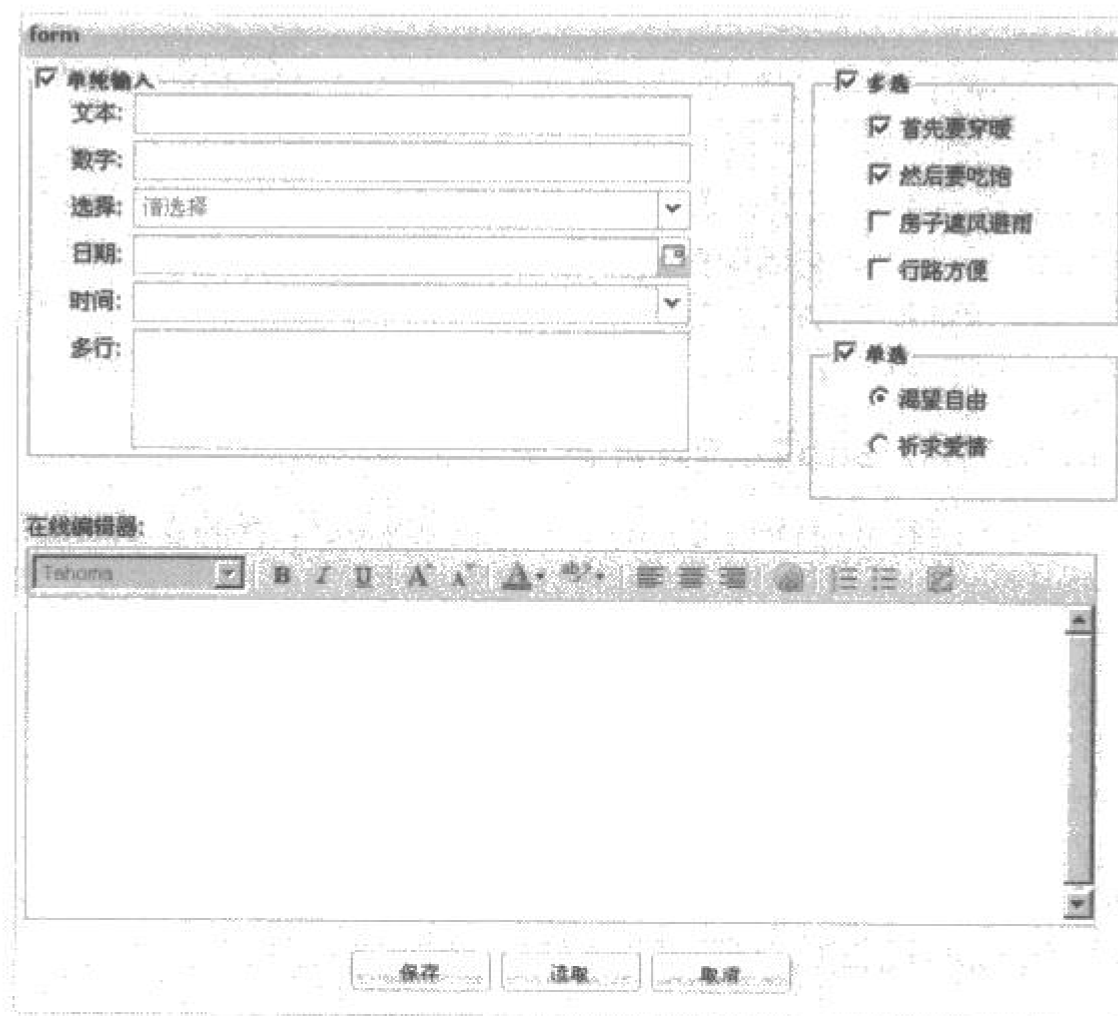


图4-3 输入控件预览

该示例的完整代码如下：

```
Ext.onReady(function(){

    // HtmlEditor需要这个
    Ext.QuickTips.init();

    var form = new Ext.form.FormPanel({
        labelAlign: 'right',
        labelWidth: 50,
        width: 600,
        title: 'form',
        frame: true,
        items: [{
            layout: 'column',
            items: [{
                columnWidth:.7,
                xtype:'fieldset',
                checkboxToggle:true,
                title: '单纯输入',
```

```
        autoHeight:true,
        defaults: {width: 300},
        defaultType: 'textfield',
        items: [{
            fieldLabel: '文本',
            name: 'text'
        },{
            xtype: 'numberfield',
            fieldLabel: '数字',
            name: 'number'
        },{
            xtype:"combo",
            fieldLabel: '选择',
            name: 'combo',
            store: new Ext.data.SimpleStore({
                fields: ['value', 'text'],
                data: [
                    ['value1', 'text1'],
                    ['value2', 'text2']
                ]
            }),
            displayField: 'text',
            valueField: 'value',
            mode: 'local',
            emptyText:'请选择'
        },{
            xtype: 'datefield',
            fieldLabel: '日期',
            name: 'date'
        },{
            xtype: 'timefield',
            fieldLabel: '时间',
            name: 'time'
        },{
            xtype: 'textarea',
            fieldLabel: '多行',
            name: 'textarea'
        },{
            xtype: 'hidden',
            name: 'hidden'
        }]
    },{
        columnWidth:.3,
        layout:'form',
        items:[{
            xtype:'fieldset',
            checkboxToggle:true,
            title: '多选',
            autoHeight:true,
            defaultType: 'checkbox',
            hideLabels: true,
            style: 'margin-left:10px;',
            bodyStyle: 'margin-left:20px;',
            items: [{
```

```
                boxLabel: '首先要穿暖',
                name: 'check',
                value: '1',
                checked: true,
                width: 'auto'
            },{
                boxLabel: '然后要吃饱',
                name: 'check',
                value: '2',
                checked: true,
                width: 'auto'
            },{
                boxLabel: '房子遮风避雨',
                name: 'check',
                value: '3',
                width: 'auto'
            },{
                boxLabel: '行路方便',
                name: 'check',
                value: '4',
                width: 'auto'
            }]
        },{
            xtype:'fieldset',
            checkboxToggle:true,
            title: '单选',
            autoHeight:true,
            defaultType: 'radio',
            hideLabels: true,
            style: 'margin-left:10px;',
            bodyStyle: 'margin-left:20px;',
            items: [{
                boxLabel: '渴望自由',
                name: 'rad',
                value: '1',
                checked: true,
                width: 'auto'
            },{
                boxLabel: '祈求爱情',
                name: 'rad',
                value: '2',
                width: 'auto'
            }]
        }]
    }]
},{
    layout: 'form',
    labelAlign: 'top',
    items: [{
        xtype: 'htmleditor',
        fieldLabel: '在线编辑器',
        id: 'editor',
        anchor: '98%'
    }]
```

```
            }],
            buttons: [{
                text: '保存'
            },{
                text: '读取'
            },{
                text: '取消'
            }]
        });

        form.render("form");

    });
```


4.3.3 基本输入控件 Ext.form.Field

Ext.form.Field是所有表单输入控件的基类，其他的输入控件都是基于Ext.form.Field扩展得来的。Ext.form.Field中定义了输入控件通用的属性和功能函数，这些通用的属性和功能函数大致分为3大类：页面显示样式、控件参数配置和数据有效性校验。

- 页面显示样式：包括clearCls、cls、fieldClass、focusClass、itemCls、invalidClass、labelStyle等属性，它们分别用来定义不同状态下输入框采用的样式。
- 控件参数配置：包括autoCreate、disabled、fieldLabel、hideLabel、inputType、labelSeparator、name、readOnly、tabIndex、value等属性，它们主要用来设置输入控件生成的DOM内容和标签内容，以及是否禁用和是否可读等配置。
- 数据有效性校验：包括invalidText、msgFx、msgTarget、validateOnBlur、validateDelay、validateEvent等属性，它们用来设置数据校验的方式以及如何显示错误信息。

我们先来看看表单输入控件可以使用的校验显示方式。默认情况下，这些输入控件会监听blur事件，如果数据校验失败就会根据msgTarget中的设置显示错误信息。通常msgTarget会被设置为'qtip'，即使用QuickTip显示错误信息，也可以将msgTarget设置为title、side、under中的一种，这样错误信息就会以指定的方式显示。因为所有的输入控件都继承自Ext.form.Field，所以我们可以为任何一个表单输入控件进行这些设置，改变它们的错误信息显示方式。

修改msgTarget的错误提示方式，如下面的代码所示。

```
Ext.QuickTips.init();

var field1 = new Ext.form.Field({
fieldLabel: 'qtip',
    msgTarget: 'qtip'
});
var field2 = new Ext.form.Field({
fieldLabel: 'title',
    msgTarget: 'title'
});
var field3 = new Ext.form.Field({
fieldLabel: 'side',
```

```
        msgTarget: 'side'
    });
    var field4 = new Ext.form.Field({
    fieldLabel: 'under',
        msgTarget: 'under'
    });

    var form = new Ext.form.FormPanel({
    title: 'form',
        frame: true,
        items: [field1, field2, field3, field4],
        renderTo: 'form'
    });
    field1.markInvalid();
    field2.markInvalid();
    field3.markInvalid();
    field4.markInvalid();
```

上面代码中的markInvalid()函数用来显示field校验出错样式。原本field需要onblur时才能进行校验并显示出错样式，在这里为了演示方便，直接使用了markInvalid()函数。图4-4演示了4种不同的错误显示方式。

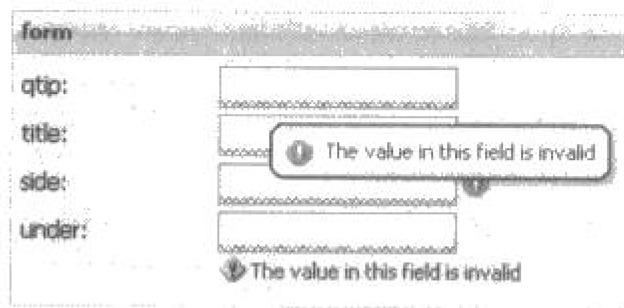


图4-4 4种错误提示效果

4.3.4 文本输入控件 Ext.form.TextField

Ext.form.TextField直接继承自Ext.form.Field，它是一个专门用来输入文本数据的输入控件。除了Ext.form.Field中提供的通用属性和功能函数外，Ext.form.TextField最常用的特性就是可以检测内部输入的数据是否为空，还可以控制输入数据的内容及最大最小长度，如下面的代码所示。

```
var field = new Ext.form.TextField({
    fieldLabel: 'empty',
    allowBlank: false,
    emptyText: '空',
    maxLength: 50,
    minLength: 10
});
```

上例中，我们为Ext.form.TextField设置了非空检测，并限制输入的字符数必须在10至50之间。其中emptyText会在输入为空时显示一个默认的提示信息，这为我们提供了良好的用户体验，如图4-5所示。



图4-5 使用emptyText设置提示信息

然后我们将上面的field放入form中，再渲染在id为form的div上。如下面代码所示：

```
var form = new Ext.form.FormPanel({
    title: 'form',
    frame: true,
    items: [field],
    renderTo: 'form'
});
```

表单控件最后都是放入表单中，然后渲染到div中的。因此下面介绍的表单控件会省略这部分代码。

4.3.5 多行文本输入控件 Ext.form.TextArea

Ext.form.TextArea也是用来输入文本的输入控件，与Ext.form.TextField的不同之处是，它可以输入多行文本。除此之外，两者的用法都是相同的。我们在使用它时只需要设置对应的长度和宽度就可以了，如下面的代码所示。

```
var field = new Ext.form.TextArea({
    width: 200,
    grow: true,
    preventScrollbars: true,
    fieldLabel: 'empty',
    allowBlank: false,
    emptyText: '空',
    maxLength: 50,
    minLength: 10
});
```

上例中将Ext.form.TextArea的宽度设置为200，对于grow:true的情况，Ext.form.TextArea会根据输入的内容自动修改自身的高度，而不是像Ext.form.TextField那样修改自身的宽度。preventScrollbars参数的用途是防止出现滚动条，如果内容超出显示范围，就会自动隐藏。

Ext.form.TextArea的显示效果如图4-6所示。

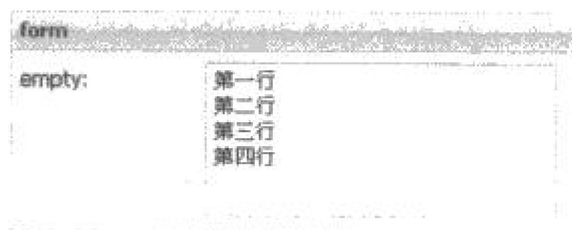


图4-6 Ext.form.TextArea根据输入数据自动调节高度

4.3.6 日期输入控件 Ext.form.DateField

Ext.form.DateField是平时经常用到的日期选择控件，除了弹出日历选择日期的功能之外，它还支持所有Ext.form.Field以及Ext.form.TextField中定义的功能，如下面的代码所示。

```
var field = new Ext.form.DateField({
    fieldLabel: '日期',
    emptyText: '请选择',
    format: 'Y-m-d',
    disabledDays: [0,6]
});
```

上例中的format参数表示如何保存并显示选中的日期，我们可以采用任意的模式显示选中的日期，disabledDays的参数值是一个数组，内部可以包括0至7的整数，它可以禁止用户选择一周内的特定日期，上例的显示效果如图4-7所示。



图4-7 禁止DateField选择每周的周六和周日

4.3.7 时间输入控件 Ext.form.TimeField

Ext.form.TimeField是用来选择时间的输入控件，它可以通过指定一天中的起始时间、结束时间以及时间间隔的方式来为用户提供可供选择的时间，如下面的代码所示。

```
var field = new Ext.form.TimeField({
    fieldLabel: '时间',
    emptyText: '请选择',
    minValue: '10:00 AM',
    maxValue: '14:00 PM',
    increment: 30
});
```

在上面的代码中，我们将起始时间设置为上午10:00点，结束时间设置为下午14:00点，时间间隔为30分钟，这样我们就得到了从10:00 AM至14:00 PM的9个时间选项。页面显示效果如图4-8所示。

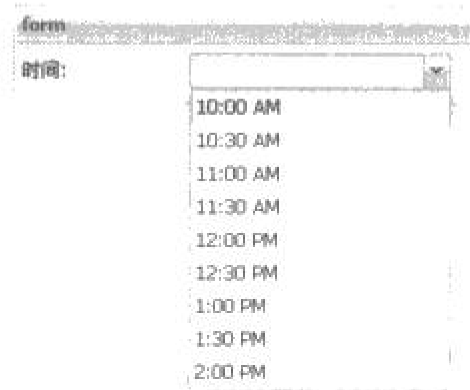


图4-8 使用TimeField设置时间的选择范围

4.3.8 在线编辑器 Ext.form.HtmlEditor

Ext.form.HtmlEditor是一个简易的在线编辑器，能对文本进行各项设置。Ext.form.HtmlEditor的页面显示样式如图4-9所示。

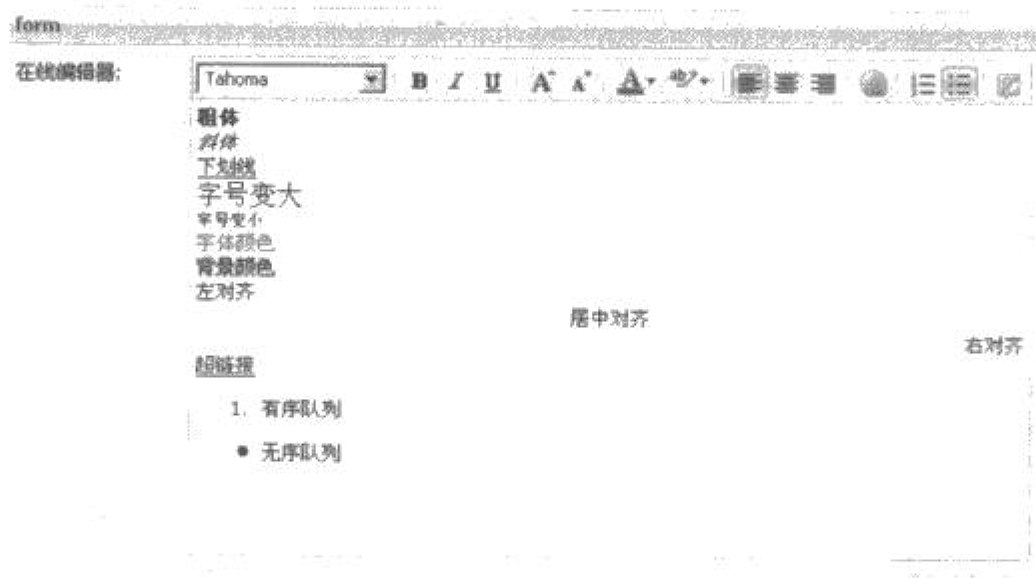


图4-9 使用在线编辑器编辑文本样式

Ext.form.HtmlEditor的使用方法如下面的代码所示。

```
var field = new Ext.form.HtmlEditor({
    fieldLabel: '在线编辑器',
    enableAlignments: true,
    enableColors: true,
    enableFont: true,
    enableFontSize: true,
    enableFormat: true,
```

```
        enableLinks: true,
        enableLists: true,
        enableSourceEdit: true
    });
```

在Ext.form.HtmlEditor中，可以用对应的enable选项启用或禁用对应的功能按钮，以此控制提供给用户使用的各种功能。

4.3.9 隐藏域 Ext.form.Hidden

Ext.form.Hidden直接继承自Ext.form.Field，可以通过它的setValue()和getValue()函数对它执行赋值和取值操作，但它不会显示在页面上。当想保存一些需要隐藏的数据时，可以使用Ext.form.Hidden，如下面代码所示。

```
var field = new Ext.form.Hidden({
    name: 'hiddenId'
});
field.setValue("some thing");
var value = field.getValue();
```

4.3.10 下拉输入框 Ext.form.TriggerField

Ext.form.TriggerField是ComboBox、DateField和TimeField的父类，它既可以手工录入数据，也可以通过选择录入数据。为了显示下拉选择的功能，我们需要覆写它的onTriggerClick()函数以实现弹出窗口，如下面的代码所示。

```
var field = new Ext.form.TriggerField({
    fieldLabel: '选择',
    name: 'name',
    onSelect: function(record){
    },
    onTriggerClick: function() {
        if (this.menu == null) {
            this.menu = selectMenu;
        }
        this.menu.show(this.el, "tl-bl?");
    }
});
```

在自定义的onTriggerClick()函数中，我们判断当前对象的menu属性是否存在，如果不存在，则把menu属性设置为selectMenu，最后将菜单显示出来，这里的tl-bl?表示弹出菜单的左上角或左下角与Ext.form.TriggerField对齐。

这里使用的selectMenu是预先创建的一个Ext.menu.Menu实例，我们为这个菜单添加了一个Ext.grid.GridPanel，并为Ext.grid.GridPanel设置了监听函数，在单击任意一行时都可以将对应行的索引值赋值给Ext.form.TriggerField，并隐藏菜单，代码如下所示。

```
grid.on('rowclick', function(grid, rowIndex, e) {
    selectMenu.hide();
    field.setValue(rowIndex);
});
```

最终的页面显示效果如图4-10所示。



图4-10　自定义TriggerField从表格中选择数据

示例的完整代码如下：

```
var grid = new Ext.grid.GridPanel({
    width: 300,
    autoHeight: true,
    title: 'grid',
    store: new Ext.data.SimpleStore({
        data: [
            ['name1','female','descn1'],
            ['name2','male','descn2']
        ],
        fields: ['name','sex','descn']
    }),
    columns: [
        {header:'姓名',dataIndex:'name'},
        {header:'性别',dataIndex:'sex'},
        {header:'备注',dataIndex:'descn'}
    ],
    viewConfig: {
        forceFit: true
    }
});

var selectMenu = new Ext.menu.Menu({
    items: [new Ext.menu.Adapter(grid)]
});

var field = new Ext.form.TriggerField({
    fieldLabel: '选择',
    name: 'name',
    onSelect: function(record){
    },
    onTriggerClick: function() {
        if (this.menu == null) {
            this.menu = selectMenu;
        }
        this.menu.show(this.el, "tl-bl?");
    }
});

grid.on('rowclick', function(grid, rowIndex, e) {
```

```
        selectMenu.hide();
        field.setValue(rowIndex);
    });

    var form = new Ext.form.FormPanel({
        title: 'form',
        frame: true,
        items: [field],
        renderTo: 'form'
    });
```

4.4 使用表单提交数据

表单最重要的功能就是向后台提交数据。但是，普通的HTML表单只能执行最单纯的提交，EXT中的表单却支持3种形式的提交：原始HTML表单形式的提交和两种Ajax形式的提交。接下来我们会详细讲解这些内容。

下面的示例是一个只包含一个TextField的表单，我们把这些数据提交到后台。

4.4.1 EXT 默认的提交形式

表单对象拥有一个submit函数，用途就是提交数据，如代码清单4-2所示。

代码清单4-2 默认提交方式

```
Ext.onReady(function(){
    var form = new Ext.form.FormPanel({
        defaultType: 'textfield',
        labelAlign: 'right',
        title: 'form',
        labelWidth: 50,
        frame:true,
        width: 220,
        url: '04_01_01.jsp',
        items: [{
            fieldLabel: '文本框',
            name: 'text'
        }],
        buttons: [{
            text: '按钮',
            handler: function() {
                form.getForm().submit();
            }
        }]
    });
    form.render("form");
});
```

这里要为form设置一个URL参数，表示表单数据将被提交到该路径。记得给TextField加上一个name属性，这样后台才知道接收到的数据来自哪个控件。最后看一下EXT中的按钮，text表示按钮显示的文字，handler则是单击按钮时调用的函数。这里直接调用表单的submit()函数，

将整个表单中的数据提交到后台。

因为FormPanel是布局容器，没有提供submit()函数，所以要先获得它内部包含的BasicForm，才能提交。

在该示例中，后台接收数据的脚本是04_01_01.jsp，与以前的方式相同，唯一的区别就是不再跳转，只需要返回一个JSON字符串提示操作是否成功，如下面的代码所示。

```
<%@ page contentType="text/html;charset=utf-8"%>
<%
    request.setCharacterEncoding("UTF-8");
    response.setCharacterEncoding("UTF-8");

    String text = request.getParameter("text");
    System.out.println(text);

    response.getWriter().print("{success:true,msg:'成功'}");
%>
```

示例见04.form/04-01-01.html和04_01_01.jsp。

解决了一个问题，另一个问题又出现了。表单默认使用Ajax提交数据，我们怎么知道提交是否成功呢？当按下“提交”按钮时没有任何反应该怎么办呢？别急，连Ajax都是可以回调的，何况表单还封装了自己的处理方式。

要想获得反馈，首先要修改submit方法，让它支持更多的功能，如下面的代码所示。

```
form.getForm().submit({
    success:function(form, action){
        Ext.Msg.alert('信息', action.result.msg);
    },
    failure:function(){
        Ext.Msg.alert('错误', '操作失败！');
    }
});
```

按下“提交”按钮，如果成功，就会出现图4-11的效果。

在上面的代码中，success和failure各自对应一个函数，与直接使用Ajax有些不同。

- 函数中传入的参数不一样：第一个参数是form，第二个参数是action。
- 触发的条件不同。

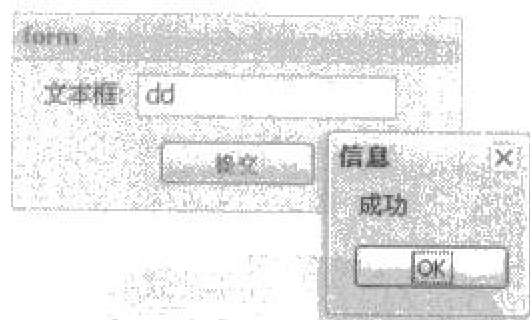


图4-11 提交成功

在Ext.lib.Ajax中，判断究竟是调用success还是failure的条件与业务无关。如果Http响应成功，就执行success；如果出现404或500错误，就执行failure。

form中的success和failure则是与业务相关的，只有后台响应为true或响应的JSON中包含success:true时，才执行success()函数。不过，这样一来failure()函数就复杂了，如何判断是连接失败，还是业务上出了问题呢？为了区分它们，EXT默认规定：如果响应的JSON中是success不是true，并且响应的JSON中包含errors:{}，那么就认为是业务错误；如果不包含errors:{}，就认为是连接失败。当业务错误时，用this.failureType = Ext.form.Action.SERVER_INVALID;标记，可以通过action.failureType进行判断。

这里就是封装的结果，第一个参数是form对象，如果想实现表单的功能，直接使用这个form参数便可。例如，想在提交一次之后清空所有数据，直接用form.reset()就行了。第二个参数是响应的信息，action.result给我们提供了一个简易通道，这样省去了先获得responseText再转换成JSON的麻烦，现在可以直接从它里边调用返回的JSON数据了。比如，后台返回的信息就可以直接通过action.result.msg获得。

示例见04.form/04-01-02.html。

4.4.2 使用 HTML 原始的提交形式

EXT默认的提交形式是不会进行页面跳转的，主要是考虑到"one page one application"（在同一个页面中实现整体应用）的形式。但是，有的用户还是希望在点击"提交"按钮后就可以跳转到其他页面。在EXT里这并不是什么难事，不过要再次强调一下，EXT的本质就是JavaScript，原来JavaScript 能做的事情它都可以做。Ext.form.Form Panel也只是对原始的FORM标签进行了装饰而已。找出那个我们都熟悉的FORM标签，直接在它上面调用submit()就可以了。

其实，难点就在于找出Ext.form.Form中层层包裹的FORM标签。注意，EXT中所有的控件都有el，el都是有DOM的，这个DOM模型就是EXT控件在页面上真实对应的部分了。于是我们只需要修改按钮的handler函数，如下面的代码所示。

```
handler: function() {
    form.getForm().getEl().dom.action = "04_01_01.jsp";
    form.getForm().getEl().dom.submit();
}
```

要注意的是，EXT动态生成了表单，却没有把action部分添加上。如果对action进行手工设置，它就只能刷新当前页面了。

单击"提交"按钮，页面就会刷新，随即变成04_01_01.jsp了。页面上会显示本来应该传给Ajax的字符串。切记，这样造成的页面跳转很可能会导致"one page one application"系统变得紊乱，还是建议在普通的环境下使用。

示例在04.form/04-02-01.html中。

4.4.3 单纯 Ajax

这里跳过表单自带的Ajax功能，使用单独的Ajax。实际上，表单自身的submit就使用了Ajax方式。不过，如果你想进一步控制表单里的数据，那么调用Ajax也是一个不错的选择。

既然是使用外部Ajax，我们就只需知道如何从中把字段的数据取出来，有以下几种方式。

- form.getValues()函数：它有一个参数，如果参数是true，就返回JSON组装的字符串；如果是false，就返回JSON对象，对应其中每个字段的名称和值。默认是false。
- findField()函数：它可以获得表单里的控件。例如，我们现在有一个TextField，名称是text，那么就可以通过下面的方式获得。

```
var text = form.getForm().findField('text');
```

我们使用最简单的getValues(true)函数来配合Ajax，用getValues(true)获得数据，然

后用Ajax传递给后台，最后判断回调，如下面的代码所示。

```
handler: function() {
    var text = form.getForm().findField('text');

    Ext.lib.Ajax.request(
        'POST',
        '04_01_01.jsp',
        {success:function(response){
            var result = Ext.decode(response.responseText);
            Ext.Msg.alert('信息', result.msg);
        },failure:function(){}},
        form.getForm().getValues(true)
    );
}
```

使用form.getValues(true)时需要注意的是，我们得到的字符串并不是使用Ext.decode()编码后的JSON格式字符串，而是name=value的形式，无需处理就可以发送给后台。不过也出现了一个问题，因为里边包含了“=”和“&”，无法使用encodeURIComponent()函数对它进行编码。这样一来，如果参数中包含中文就不能使用GET发送数据了，否则会出现乱码，实际应用中还需要权衡使用。

示例见04.form/04-03-01.html。

4.5 数据校验

数据校验是非常必要的，因为用户输入的数据有时候是不可靠的。如果不能对用户输入的数据进行校验和处理，也许开发出来的应用根本无法使用。必须先规定好各项数据该如何填写，如果输入错误就不能提交到后台。关于这一点，IE和Firefox有些不同。如果校验失败，在Firefox下调用submit()是不会执行提交的。但是，在IE下必须先使用form.isValid()自行判断，如果返回false，就不能再调用submit()，否则仍然会将非法数据提交到后台。

EXT把验证封装到了表单的控件中，下面让我们来看看如何使用它们。

4.5.1 输入不能为空

最基本的验证就是文本框或其他控件中必须输入值，如下面的代码所示。

```
new Ext.form.TextField({
    name: 'text',
    fieldLabel: '文本框',
    allowBlank: false
});
```

allowBlank默认为true，也就是说可以不输入数据，将它改成false，再看看会出现什么效果，如图4-12所示。

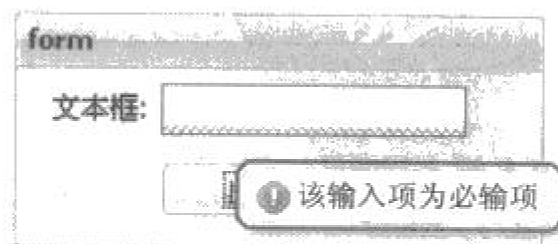


图4-12 必输项

如果没有输入任何数据，文本框下就会出现红线，图4-12中鼠标放上去出现提示的功能需要Ext.

QuickTips.init()的支持。

其实这种提示方式比弹出提示对话框更好用，也更容易让用户接受。不过，因为allowBlank不是在Ext.form.Field中定义的，所以并不是所有控件都可以使用它。allowBlank最早出现在Ext.form.TextField中，只有继承它的控件才可以使用非空校验，具体可参考图4-13中的继承树。

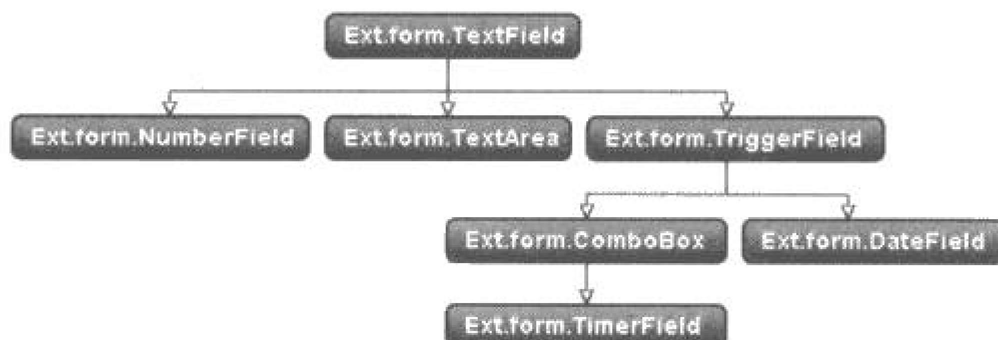


图4-13 支持allowBlank的组件继承树

示例在04.form/05-01-01.html中。

4.5.2 最大长度和最小长度

限制输入的最大长度是一种常见的校验方式，数据库只允许输入255个字符。当用户输入的数据超出最大长度时就会引起错误。允许输入的最小长度校验刚好相反，故在此一并介绍。

如下面的代码所示：

```
new Ext.form.TextField({
    fieldLabel: '文本框',
    name: 'text',
    maxLength: 10,
    minLength: 5
});
```

我们设置最大长度不能超过10个字符，最小长度不能少于5个字符，如果输入的字符数量大于10，便会出现图4-14中的提示。



图4-14 最大长度校验

示例在04.form/05-02-01.html中。

4.5.3 借助 vtype

EXT提供了一套默认的校验方案，其实就是一系列输入规则和错误提示。如果使用它们，就不需要再去背诵那一长串正则表达式，只需记住vtype的值，然后配置在控件中即可，如下面的代码所示。

```
new Ext.form.TextField({
    fieldLabel: '文本框',
```

```
    name: 'text',
    vtype: 'email'
})
```

这里我们给vtype设置的是e-mail，限制text只能使用邮箱地址的格式，非常简单。

这些验证信息都定义在Ext.form.VTypes中，默认情况下，VTypes一共有4种验证信息，如下所述。

- alpha：只能输入英文字母。
- alphanum：只能输入字母和数字。
- email：电子邮箱。
- url：网址。

需要使用哪一种验证信息，只需要把vtype设置成对应的值即可。当然也可以自己进行扩展，具体的方式可以参照VTypes.js。

示例在04.form/05-03-01.html中。

4.5.4 自定义校验规则

其实自定义校验规则就是允许自定义正则表达式。可以编写一个regex对输入数据进行校验，这样一来，无论是何种形式的数据，都能对其进行判断。如果你对regex不了解或没有写过正则表达式，可以跳过本节，因为篇幅原因，这里不会讲解如何使用正则表达式。

这里，使用regex验证只能输入汉字的情况，如下面的代码所示。

```
new Ext.form.TextField({
    fieldLabel: '文本框',
    name: 'text',
    regex: /^[\u4E00-\u9FA5]+$/,
    regexText: '只能输入汉字'
})
```

效果如图4-15所示。其实原理很简单，就是为regex设置一个正则表达式，然后在进行校验时会调用regex.test(value)。如果为true，就表示校验通过；如果为false，就提示校验失败，同时在提示框中显示regexText的内容。

示例在04.form/05-04-01.html中。



图4-15 regex校验

4.5.5 算不上校验的NumberField

Ext.form.NumberField是Ext.form.TextField的子类，它本来应该继承上面所讲述的那些验证方式。但是，NumberField却有更特殊的验证方式，所以在此单独讲解一下。

在NumberField中只能输入数字，这种校验方式是其他控件都没有的。其他控件都是在提交时对已输入的数据进行校验。NumberField却根本不让输入不符合要求的数据。下面检测一下这种方式的校验强度。

- 直接从键盘输入字母和非数字字符：无法输入NumberField。

- 尝试输入有多个小数点的非法数字字符：NumberField只能限制输入的字符，一旦输入结束，它就会自动进行校验。从左侧开始截取，一直保留到合法的数字格式为止。例如，1.0.0.0就变成了1.00。
- 直接把带有非数字的字符串粘贴到NumberField：粘贴是成功了，但在编辑结束后，NumberField又会对内容进行校验，把非法数据转化成最接近的数字。

由此可见，不管输入什么样的数据，它最后只保留有效的数字，必须承认EXT实在是考虑得非常周到。下面再接着讲解与数字配置相关的一些内容。

- 不想让用户输入负数：将参数allowNegative设置为false，于是用户不能再输入负号，参数的默认值为true。
- 不想让用户输入小数：将参数allowDecimals设置为false，于是用户不能输入小数点，参数的默认值为true。
- 设置小数精度：默认情况下是保留到小数点后两位，即百分位。修改参数decimalPrecision的值，决定精确到小数点后多少位。
- minValue和maxValue规定可输入数字的范围：这是数字专有的，比如一个人的年龄输入值在1到150之间。

不过，这还是有点问题，即使设置了allowNegative:false和allowDecimals:false，也可以输入负号和小数点。虽然这两个符号会在验证时自动消失，但还是会让人觉得难以接受。maskRe参数的用途和NumberField一样，都可以实现禁止用户输入不符合标准的数据。

如果使maskRe等于/\d/，它也是指向了一个正则表达式，把它放入NumberField里就可以阻止符号和小数点，以上的几种校验方式如下面代码所示：

```
items: [{
    fieldLabel: '数字',
    name: 'text',
    xtype: 'numberfield',
    allowNegative: false,
    allowDecimals: false,
    decimalPrecision: 4,
    minValue: 0,
    maxValue: 150,
    maskRe: /\d/
}]
```

示例在04.form/05-05-01.html中。

4.5.6 使用后台返回的校验信息

这里用到的就是前面说的第一种提交方式，在返回的响应中可以包含校验失败的信息。只要你写对了地方，表单就会自己处理它们，还会把它们放在正确的地方，如代码清单4-3所示。

代码清单4-3 使用后台返回的校验信息

```
items: [{
    fieldLabel: '文字一',
    name: 'text1'
```

```
    },{
        fieldLabel: '文字二',
        name: 'text2'
    }],
    buttons: [{
        text: '按钮',
        handler: function() {
            form.getForm().submit({
                success:function(form, action){
                    Ext.Msg.alert('信息', action.result.msg);
                },
                failure:function(form, action){
                    if (action.failureType == Ext.form.Action.SERVER_INVALID) {
                        Ext.Msg.alert('错误', '后台校验失败');
                    } else {
                        Ext.Msg.alert('错误', '无法访问后台');
                    }
                }
            });
        }
    }]
```

上面的表单中有两个文本框：text1和text2。

为了配合后台校验，我们在submit的回调函数failure()中做了点儿工作，通过action的failureType来判断响应失败是因为校验失败还是因为进行HTTP连接时发生了错误。从这里我们可以了解表单中的提交与普通Ajax的区别，普通Ajax的failure()回调函数只在发生HTTP连接错误时才会执行，而表单的failure()回调函数可以处理包括连接错误和后台业务错误在内的两种情况。

现在我们看看校验失败时后台响应的信息：

```
{success:false,errors:{text1:'有问题一',text2:'有问题二'}}
```

响应中的success:false表示后台业务失败，failure()回调函数会在出现success:false时执行。

实现后台数据校验的关键部位在errors上，它对应的是一个JSON对象，里边包含的就是错误信息，其中的text1和text2分别对应了表单中两个文本框的校验信息。表单会自动将校验信息与对应的文本框结合起来，最后在页面中显示的结果如图4-16所示。

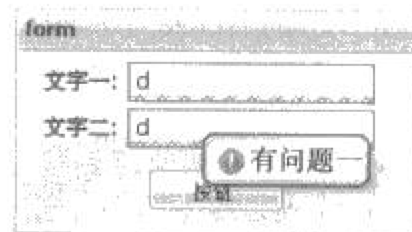


图4-16　后台校验信息

示例在04.form/05-06-01.html中。

4.6　表单布局

对表单来说，布局也是比较重要的一部分，表单是用来录入数据的。通常我们都会采用各种布局让表单元素和表单搭配得更整洁、更美观，如图4-17所示。下面我们来讨论一下有关表单的布局问题。



图4-17　默认布局

4.6.1 默认的平铺布局

如果不进行任何设置，而是直接把控件加入表单中，使用的就是自上而下的默认布局（见图4-17）。控件部分的实现代码如下所示：

```
items: [
    {fieldLabel: '俩字'},
    {fieldLabel: '三个字'},
    {fieldLabel: '四个汉字'}
],
```

表单的默认布局的优点是简便和无需额外配置，直接使用默认布局也能满足大多数要求，我们下面再来看看与其相关的其他样式配置。

你注意到了这3个输入框（见图4-17）的标签文字长度有所不同了吗？表单中的标签默认使用左对齐的方式，一共有left、top、right3个值可以选，我们可以通过配置labelAlign: 'right'使标签右对齐。标签文字的宽度也可以设置，我们这里使用labelWidth: 60，宽度就是60像素。如果标签宽度过长，文字会自动换行。按钮的位置也可以设置，有left、center、right3种选择，默认是right（右对齐）。代码如下所示：

```
defaultType: 'textfield',
labelAlign: 'right',
title: 'form',
labelWidth: 60,
```

假设我们设置了labelAlign: 'left'、labelWidth: 40和buttonAlign:'right'，表单的显示效果如图4-18所示。

示例在04.form/06-01-01.html和06-01-02.html中。

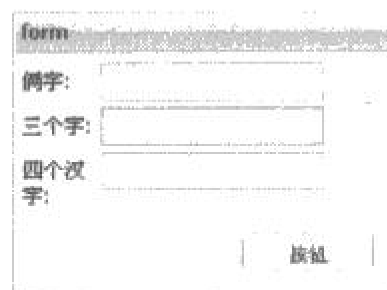


图4-18 设置标签宽度

4.6.2 平行分列布局

在图4-19中，表单中的文本框被分为3列显示。



图4-19 平行分列布局

我们先使用layout: 'column'来说明下面要使用的是列布局，然后在items指定的每列中使用columnWidth指定每列所占总宽度的百分比。

要注意一点，如果使用了分列布局，就不能在表单中直接使用defaultType指定默认的xtype了，否则会影响布局结果。每一列中也必须手动指定使用layout: 'form'，这样才能在每列中正常显示输入框和对应标签。顺便说一下，layout: 'form'其实就是FormPanel默认使用的布局方式，即自上而下依次排列（见代码清单4-4）。

代码清单4-4 分列布局

```
var form = new Ext.form.FormPanel({
    labelAlign: 'right',
    labelWidth: 60,
    title: 'form',
    frame:true,
    width: 650,
    url: '04_01_01.jsp',

    items: [{
        layout:'column',
        items: [{
            columnWidth:.33,
            layout: 'form',
            items:[{xtype: 'textfield',fieldLabel: '俩字'}]
        },{
            columnWidth:.33,
            layout: 'form',
            items:[{xtype: 'textfield',fieldLabel: '三个字'}]
        },{
            columnWidth:.33,
            layout: 'form',
            items:[{xtype: 'textfield',fieldLabel: '四个汉字'}]
        }]
    }],
    buttons: [{
        text: '按钮',
        handler: function() {
            form.getForm().submit();
        }
    }]
});
```

这里从最顶级分成3列，每列占总宽度的1/3。每列都使用表单布局，每列都只包含一个TextField，这就完成了我们上面看到的3列布局了。

示例在04.form/06-02-01.html中。

如果在每列中放多个输入框（见图4-20），只需将上面例子中fieldLabel增加到多个即可，修改之后代码如下：

```
items: [{
    layout:'column',
    items: [{
        columnWidth:.33,
        layout: 'form',
        defaultType: 'textfield',
        items:[
            {fieldLabel: '俩字'},
            {fieldLabel: '俩字'}
        ]
    },{
        columnWidth:.33,
        layout: 'form',
```

```
        defaultType: 'textfield',
        items:[
            {fieldLabel: '三个字'},
            {fieldLabel: '三个字'},
            {fieldLabel: '三个字'}
        ]
    },{
        columnWidth:.33,
        layout: 'form',
        defaultType: 'textfield',
        items:[
            {fieldLabel: '四个汉字'},
            {fieldLabel: '四个汉字'},
            {fieldLabel: '四个汉字'},
            {fieldLabel: '四个汉字'}
        ]
    }]
}],
```

图4-20 分列布局：每列中放入多个输入框

示例在04.form/06-02-02.html中。

下面介绍分列布局与默认布局相结合的情况，图4-21中的效果就是在整体上使用了默认布局，在上部使用了分列布局。

图4-21 分列布局与默认布局相结合

这里不必使用CSS中的clear:true指定换行，在layout: 'column'下直接放一个textarea即可。综合运用表单布局和分列布局，可以实现各种复杂的布局效果（如代码清单4-5所示）。

代码清单4-5　综合运用表单布局和分列布局实现复杂的布局效果

```
items: [{
    layout:'column',
    items: [{
        columnWidth:.5,
        layout: 'form',
        defaultType: 'textfield',
        items:[
            {fieldLabel: '俩字'},
            {fieldLabel: '俩字'}
        ]
    },{
        columnWidth:.5,
        layout: 'form',
        defaultType: 'textfield',
        items:[
            {fieldLabel: '三个字'},
            {fieldLabel: '三个字'},
            {fieldLabel: '三个字'}
        ]
    }]
},{
    width: 345,
    height: 100,
    xtype: 'textarea',
    fieldLabel: '四个汉字'
}],
```

示例就在04.form/06-02-03.html中。

4.6.3　在布局中使用 fieldset

在标准HTML中，需要把输入项都放到fieldset中，以此来显示分组结构。虽然EXT中的表单已经很漂亮了，但我们依然可以用fieldset来进行内部分组。

为了突出显示效果，这里我们把column和fieldset（fieldset只是一个普通的xtype）结合在一起使用（见代码清单4-6）。

代码清单4-6　使用fieldset

```
items: [{
    layout:'column',
    items: [{
        columnWidth:.5,
        layout: 'form',
        xtype: 'fieldset',
        title: '俩字',
        autoHeight: true,
        defaultType: 'textfield',
        items:[
            {fieldLabel: '俩字'},
            {fieldLabel: '俩字'}
```

```
            ]
        },{
            columnWidth:.5,
            layout: 'form',
            xtype: 'fieldset',
            title: '三个字',
            autoHeight: true,
            style: 'margin-left: 20px;',
            defaultType: 'textfield',
            items:[
                {fieldLabel: '三个字'},
                {fieldLabel: '三个字'},
                {fieldLabel: '三个字'}
            ]
        }]
    },{
        xtype: 'fieldset',
        title: '四个汉字',
        autoHeight: true,
        items: [{
            width: 345,
            height: 100,
            xtype: 'textarea',
            fieldLabel: '四个汉字'
        }]
    }],
```

注意加上标题title并设置autoHeight:true，让fieldset看起来更漂亮一些。style: 'margin-left: 20px;'则是我们直接设置的CSS样式，这是为了使第二列和第一列不会靠得太近，最后的效果如图4-22所示。

图4-22　在布局中使用fieldset

不用去记忆大量的复杂函数的用法，但仍然可以使用EXT提供的布局功能实现漂亮的效果。示例在04.form/06-03-01.html中。

4.6.4 在 fieldset 中使用布局

直接在表单里使用分列布局很简单，相信熟悉EXT的读者都会使用。然而，却有不少读者不会在fieldset中使用分列布局，包括一些对EXT比较熟悉的读者。如果按照使用表单的方法来使用它，可能会看不到fieldLabel的值，需要经过多次调试才能成功（见代码清单4-7）。

代码清单4-7　在fieldset中使用分列布局

```
set = new Ext.form.FieldSet({
    title:'fieldset',
    //width:400,
    height:80,
    columnWidth: 1,
    layout: 'column',
    border: true,
    anchor:'100%',
    labelWidth: 40,
    items:[{
        columnWidth:.4,
        layout: 'form',
        border:false,
        items: [{
                  xtype:'textfield',
                  fieldLabel: 'aaaaa',
                  name: 'aaa',
                  anchor: '95%'
        }]
    },{
        column Width:.4,
        layout: 'form',
        border:false,
        items: [{
            xtype:'textfield',
            fieldLabel: 'bbbbb',
            name: 'bbb',
            anchor:'95%'
        }]
    },{
        column Width:.2,
        layout: 'form',
        border:false,
        items: [{
            xtype:'button',
            text:'查询',
            iconCls: 'query',
            handler: function(){
                Ext.Msg.alert('sss','aaa');
            },
            scope: this
```

```
            }]
        }]
    });
    setform = new Ext.form.FormPanel({
        height: 80,
        border: false,
        labelWidth:80,
        labelAlign:'right',
        frame:true,
        items:[set]
    });

    win = new Ext.Window({
        title: 'FieldSet的column布局',
        layout:'fit',
        width:500,
        height:300,
        closeAction:'hide',
        items: [setform]
    });
    win.show();
```

这样就实现了fieldset的分列布局，效果如图4-23所示。

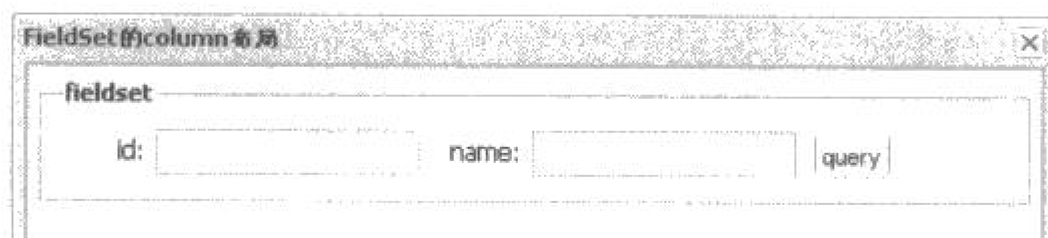


图4-23　在fieldset中使用分列布局

说明　表格布局的字体与其他布局的字体不同，还需要使用CSS控制字体。分列（column）布局与表格（table）布局都可以实现横向按列布局，具体使用时要看个人喜好和习惯了。

示例在04.form/06-04-01.html中。

4.6.5　自定义布局

下面我们看看如何向表单中添加不属于Ext.form.Field子类的控件，比如图片和文字之类的静态内容。

因为Ext.form.FormPanel继承自Ext.Panel，所以可以使用layout和items提供各种内部布局形式。除了Ext.form.Field之类的输入控件外，还可以使用其他Panel来装饰表单。这里我们就使用xtype:'panel'好了，在它里边使用img来显示图片（见代码清单4-8）。

代码清单4-8　在表单内使用Ext.Panel

```
{
    columnWidth:.5,
    layout: 'form',
    xtype: 'fieldset',
    title: '三个字',
    autoHeight: true,
    style: 'margin-left: 20px;',
    defaultType: 'textfield',
    items:[
        {fieldLabel: '三个字'},
        {fieldLabel: '三个字'},
        {fieldLabel: '三个字'},
        {xtype: 'panel',html: '<center><img src="user_female.png" /></center>'}
    ]
}
```

运行效果如图4-24所示。

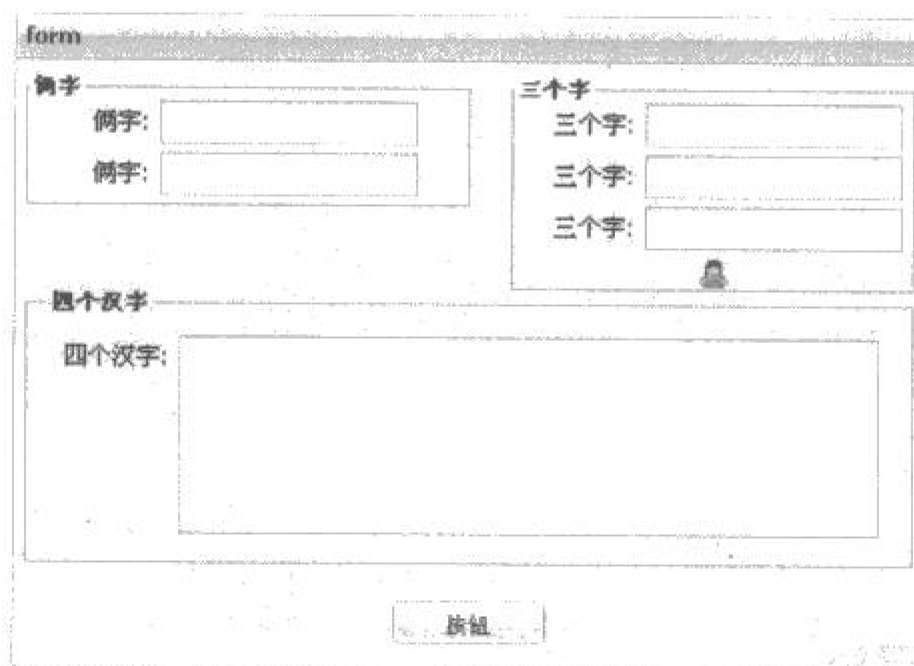


图4-24　向Ext.form.FormPanel中添加Ext.Panel

表单的完整代码如下所示：

```
var form = new Ext.form.FormPanel({
    labelAlign: 'right',
    labelWidth: 60,
    title: 'form',
    frame:true,
    width: 450,
    url: 'form.jsp',

    items: [{
        layout:'column',
```

```
        items: [{
            columnWidth:.5,
            layout: 'form',
            xtype: 'fieldset',
            title: '俩字',
            autoHeight: true,
            defaultType: 'textfield',
            items:[
                {fieldLabel: '俩字'},
                {fieldLabel: '俩字'}
            ]
        },{
            columnWidth:.5,
            layout: 'form',
            xtype: 'fieldset',
            title: '三个字',
            autoHeight: true,
            style: 'margin-left: 20px;',
            defaultType: 'textfield',
            items:[
                {fieldLabel: '三个字'},
                {fieldLabel: '三个字'},
                {fieldLabel: '三个字'},
                {xtype: 'panel',html: '<center><img src="user_female.png" /></center>'}
            ]
        }]
    },{
        xtype: 'fieldset',
        title: '四个汉字',
        autoHeight: true,
        items: [{
            width: 345,
            height: 100,
            xtype: 'textarea',
            fieldLabel: '四个汉字'
        }]
    }],
    buttons: [{
        text: '按钮',
        handler: function() {
            form.getForm().submit();
        }
    }]
});
```

示例在04.form/06-05-01.html中。

4.7 ComboBox 详解

EXT中提供的ComboBox与HTML中原有的Select无任何关系，它完全是用div重写的。

4.7.1　ComboBox 简介

耳听为虚，眼见为实，先看看所谓的ComboBox究竟是个什么模样（见图4-25）。

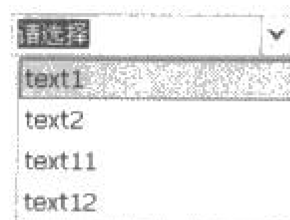


图4-25　ComboBox

是不是觉得它比原生的Select更漂亮？下面我们来看看如何制作ComboBox（见代码清单4-9）。

代码清单4-9　实现ComboBox

```
var data = [
    ['value1','text1'],
    ['value2','text2'],
    ['value11','text11'],
    ['value12','text12']
];

var store = new Ext.data.SimpleStore({
    fields: ['value', 'text'],
    data : data
});

var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '请选择',
    mode: 'local',
    triggerAction: 'query',
    valueField: 'value',
    displayField: 'text',
applyTo: 'combo',
value: 'text1'
});
```

首先，定义ComboBox中将要显示的数据，我们这里使用的是二维数组data。

其次，将这个二维数组交给Ext.data.SimpleStore。Ext.data.SimpleStore的功能与Ext.data.Store相近，而Ext.data.SimpleStore不必定义proxy和reader就可以直接使用数组，使用起来更加方便。

最后，调用applyTo('combo')把ComboBox画到页面上。需要注意的是，id="combo"对应的必须是<input id="combo" type="text" />，不能在DIV标签上渲染ComBox。

下面我们来看看Ext.form.ComboBox中用到的参数，如下所示。

- store：用来为ComboBox提供数据，原始数据是一个二维数组，将数组放入SimpleStore之后，第一列对应属性名为value，第二列对应属性名为text。
- 现在可以看到另外两个参数：valueField和displayField。
  为什么它们的值与store中定义的两个名字一样呢？它们之间有何关系？
  果然，ComboBox正是根据它们之间的对应关系来显示数据的。
  单击ComboBox，弹出的列表里就是SimpleStore中text列对应的数据。而当你选中某个数据时，ComboBox的值就被自动设置成SimpleStore中被选中那一行对应的value值，

于是数据就关联在一起了。

- emptyText很好理解，即没有选择任何数据时ComboBox里显示的提示信息。
- mode设置成local，这也就告诉ComboBox，它需要的数据已经读取到本地了，不需要再去后台读取了。
- triggerAction设置成all，如果用默认的query，它会使用autocomplete功能，autocomplete的使用会在后面详细介绍。
- value可以为ComboBox设置默认值。

如果你不了解autocomplete，可以参考图4-26中的效果。autocomplete就像下面这样，会根据输入的信息自动从结果中选择匹配的信息进行显示。

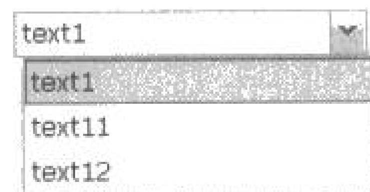


图4-26 自动匹配

有没有发现text2不见了？autocomplete会根据输入的text1将不匹配的结果过滤掉。如果想让ComboBox每次都显示所有可选数据，那么把triggerAction: 'query'换成triggerAction: 'all'即可。

示例在04.form/07-01-01中。

4.7.2 将 Select 转换成 ComboBox

这种创建ComboBox的方式要求HTML里必须先包含一个Select下拉框，如下面的代码所示。

```
<select id="combo">
    <option value="value1">text1</option>
    <option value="value2">text2</option>
    <option value="value11">text11</option>
    <option value="value12">text12</option>
</select>
```

我们在这个Select的基础上构造ComboBox，使用Select里包含的数据，如下面的代码所示。

```
var combo = new Ext.form.ComboBox({
    emptyText: '请选择',
    mode: 'local',
    triggerAction: 'all',
    transform: 'combo'
});
```

这里的代码明显比上面的不通过Select而直接创建ComboBox的代码少了很多，关键在于transform:'combo'参数，它告诉EXT把指定的Select中的数据逐条抽取出来，添加到ComboBox里，最后把Select完全替换成ComboBox。

示例在04.form/07-02-01.html中。

4.7.3 ComboBox 结构详解

首先，用Firebug查看一下生成后的HTML（见图4-27），这可以更容易明白ComboBox的绚丽外表下隐藏着的内容。

```
<div id="ext-gen7" class="x-form-field-wrap" style="width: 144px;">
    <input id="combo" class="x-form-text x-form-field
    x-form-empty-field" type="text" autocomplete="off"/>
    <img id="ext-gen8" class="x-form-trigger
    x-form-arrow-trigger" src="../../resources/images/default/s.gif"/>
</div>
<div id="ext-gen13" class="x-layer x-combo-list" style="position: absolute; z-index:
11000; visibility: hidden; left: -10000px; top: -10000px; width: 142px;">
    <div id="ext-gen15" class="x-combo-list-inner" style="width: 142px;">
        <div class="x-combo-list-item">text1</div>
        <div class="x-combo-list-item">text2</div>
        <div class="x-combo-list-item">text11</div>
        <div class="x-combo-list-item">text12</div>
    </div>
</div>
```

图4-27　ComboBox对应的DOM代码

原来就是用div包裹了一个input和img。单击显示出来的列表也是用div制作出来的。平时它会隐藏起来，当用户选择时才显示到对应的input下面，原来ComboBox就是这么简单。

实际上，ComboBox还有另外一种特性，我们只需要为它加上一个hiddenName属性。注意，这里的hiddenName不能和ComboBox的id重复，这样生成的ComboBox从外表上看不出有什么变化，但Firebug会将它的内在变化完全展示出来，如下面的代码所示。

```
var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '请选择',
    mode: 'local',
    triggerAction: 'query',
    valueField: 'value',
    displayField: 'text',
    hiddenName: 'comboId'
});
```

看到了吧？在最后一个参数中，我们将hiddenName设置为'comboId'（见图4-28），不让它和div中的id="combo"重复。

```
<div id="ext-gen7" class="x-form-field-wrap" style="width: 144px;">
    <input id="comboId" type="hidden" name="comboId" value=""/>
    <input id="combo" class="x-form-text x-form-field
    x-form-empty-field" type="text" autocomplete="off"/>
    <img id="ext-gen8" class="x-form-trigger
    x-form-arrow-trigger" src="../../resources/images/default/s.gif"/>
</div>
```

图4-28　hiddenName

从Firebug展示的DOM树中你又看到了什么？用蓝色标记出的这一行（第2行）就是使用hiddenName的结果。这让ComboBox又增加了一个type="hidden"的input，而<input type="hidden">隐藏域的id和name都为comboId（实际上，你也可以分别设置它的id和name，

用 hiddenId和hiddenName指定）。

<input type="hidden">隐藏域的value会随着你的选择而改变，它的值一直等于选中部分对应的value。

EXT之所以为ComboBox生成如此复杂的DOM结构，主要还是为了模拟HTML中的原生SELECT控件。在原生SELECT控件中，我们可以为value和text设置不同的值，用户选择的是text，而向后台提交的数据则是text对应的value。如果没有为ComboBox设置hiddenName，ComboBox提交的永远都是用户看到的text。通过使用hiddenName，我们才可以向后台提交text对应的value的值。

4.7.4 ComboBox 读取远程数据

使用远程数据其实非常简单。之前我们使用Ext.data.SimpleStore从本地数组中获得数据，这次模仿表格的方式，用Ext.data.Store配合proxy和reader获得从后台返回的数据，如下面的代码所示。

```
var store = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url:'07-04-01.txt'}),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'value'},
        {name: 'text'}
    ])
});
```

我们使用HttpProxy访问07-04-01.txt，并使用ArrayReader把获得的数据分成value和text两列，07-04-01.txt的内容如下所示。

```
[
    ['value1','text1'],
    ['value11','text11'],
    ['value111','text111'],
    ['value1111','text1111'],
    ['value11111','text11111'],
    ['value2','text2'],
    ['value22','text22'],
    ['value222','text222'],
    ['value2222','text2222'],
    ['value22222','text22222']
]
```

现在可以更新页面了。但是，为什么单击并选择后却没有显示数据呢？再次单击，还是没数据。不要着急，我们还需要配置一些参数。

我们需要把ComboBox的mode参数从local改成remote，实际上mode的默认值就是remote，如下面的代码所示。

```
var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '请选择',
    mode: 'remote',
    triggerAction: 'all',
```

```
        valueField: 'value',
        displayField: 'text'
    });
```

除了mode改成remote了以外，其他地方无需更改。

也有不修改mode:remote的方法，在创建ComboBox后手动运行store.load()即可（见代码清单4-10）。

代码清单4-10　远程加载数据

```
var store = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url:'07-04-01.txt'}),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'value'},
        {name: 'text'}
    ])
});
store.load();
var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '选择',
    mode: 'local',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text'
});
```

这种方法与上面的方法的不同之处在于，你可以决定何时对ComboBox的数据进行初始化，如果不手工运行store.load()，会在第一次单击“选择”按钮时读取数据。这两种方法的最终效果是一样的。

有一点要特别注意，如果mode设置为remote，又使用了store.load()，store就会读取两次数据，执行store.load()时一次，单击“选择”时一次。

示例在04.form/07-04-01.html中。

4.7.5　ComboBox 的高级配置

ComboBox的高级配置如下。

- 为ComboBox添加分页功能（见图4-29）。

  实际上只添加了两个参数，如下面的代码所示。

```
var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '请选择',
    mode: 'remote',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text',
    minListWidth: 220,
    pageSize: 5
});
```

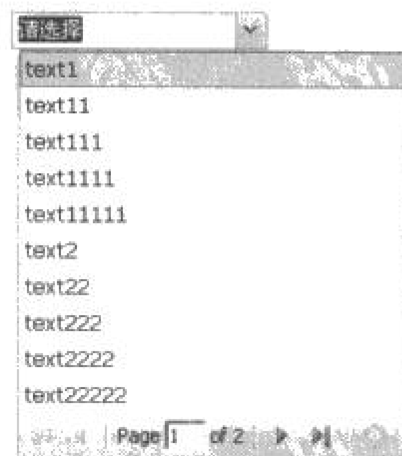


图4-29　分页

pageSize是主要参数，它决定每次显示多少条记录，EXT内部自动计算是否进行分页。minListWidth用来控制下拉列表的宽度，如果不设置它，也许就看不到完整的分页条。还有一点要注意，参数mode的值必须是'remote'。如果写成local，分页条就无法使用，因为分页的前提是先到store中分批获取数据。如果数据都已经保存到本地，就不需要执行分页操作了。

满足了上述条件之后，我们就获得了一个闪亮的分页条，现在可以使用它分页查看数据了。

- 通过拖放改变弹出列表的大小。

如果构造ComboBox时加上参数resizable:true，就可以对弹出列表进行拖放，从而修改列表的大小，如下面的代码所示。

```
var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '请选择',
    mode: 'remote',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text',
    minListWidth: 220,
    pageSize: 5,
    resizable: true
});
```

具体效果如图4-30所示，注意下拉列表右下角的拖放标志，把鼠标放到上面就可以执行拖放操作。

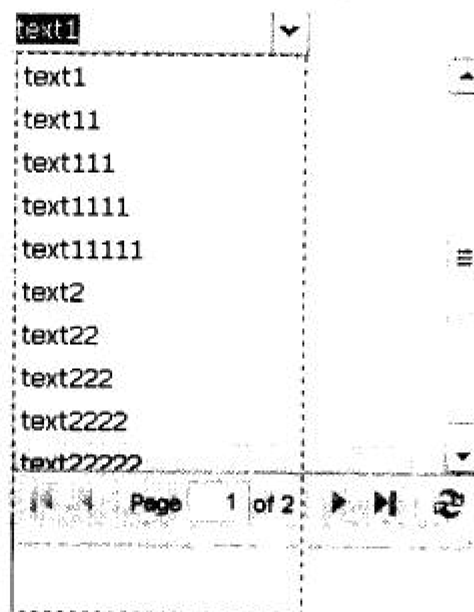


图4-30 通过拖放改变弹出列表的大小

- 是否允许用户自己填写内容。

不知道你是否发现ComboBox与原生的Select之间有一个很大的区别。ComboBox可以让用户自由填写数据。通过对ComboBox内部的分析可以看出，显示框只是一个type="text"的input输入框。当然，它是允许用户输入的，但这可能不是我们想要的效果。如何才能实现与原生Select一样的只能选择不可修改的效果呢？

我们需要的只是一个readOnly参数，如下面的代码所示。

```
var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '请选择',
    mode: 'remote',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text',
    minListWidth: 220,
    pageSize: 5,
    resizable: true,
    readOnly: true
});
```

这样用户就只能从我们提供的数据中进行选择，不能随便填写自己的数据，保持了原生Select的原貌。

下面我们将以上的3个额外配置的功能组装在一起，完整代码如下：

```
var store = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url:'07-04-01.txt'}),
    reader: new Ext.data.ArrayReader({}, [
        {name: 'value'},
        {name: 'text'}
    ])
});
store.load();

var combo = new Ext.form.ComboBox({
    store: store,
    emptyText: '请选择',
    mode: 'remote',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text',
applyTo: 'combo',
    minListWidth: 220,
    pageSize: 5,
    resizable: true,
    readOnly: true
});
```

示例在04.form/07-05-01.html中。

4.7.6　监听用户选择的数据

下面我们来讨论一下如何使用EXT中提供的事件机制监听ComboBox的事件，从而获知用户选择了哪条数据。

在用户每选择一项时，就弹出一个窗口显示用户选择的项的详细情况，如图4-31所示。

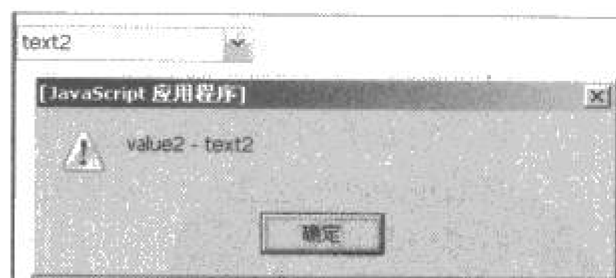


图4-31　根据用户选择的项弹出相应的信息提示窗口

你可能会觉得奇怪，我们是怎么实现这种效果的呢？EXT会在你执行某种操作时执行一些预定义的操作，这些预定义的操作就叫做事件监听，如下面的代码所示。

```
combo.on("select", function(comboBox){
    alert(comboBox.getValue() + '-' + comboBox.getRawValue());
});
```

这个on告诉ComboBox，要监听ComboBox，并设置自己的事件监听器。第一个参数'select'告诉ComboBox要监听哪个事件，第二个参数是处理这个事件的事件监听器，传入的参数是被监听的ComboBox本身。这里，我们在ComboBox发生select事件时通过getValue()和getRawValue()获得用户选定的项的具体信息。

你可能会问："你怎么知道on的第一个参数是'select'呢？"其实EXT中的每个组件支持的事件都可以从它自带的API文档中找到。打开文档中Ext.form.ComboBox页，在Public Events一栏中可以看到很长的一串事件列表，其中就包括我们刚才用到的select。

当用户选中ComboBox下拉列表中的某一项时，这个事件就会被触发。

EXT的事件机制就是如此神奇，在选中ComboBox列表中的一项后，EXT就会找到所有绑定到select事件的监听器。例如，我们刚才就用on方法把自定义的函数绑定到了select事件上。注意，你可以绑定一个函数，也可以绑定多个函数，EXT会执行所有这些函数，并把定义好的3个参数传递过去，供我们使用。完整代码如下所示：

```
var data = [
        ['value1','text1'],
        ['value2','text2'],
        ['value11','text11'],
        ['value12','text12']
    ];

    var store = new Ext.data.SimpleStore({
        fields: ['value', 'text'],
        data: data
    });

    var combo = new Ext.form.ComboBox({
        store: store,
        emptyText: '请选择',
        mode: 'local',
        triggerAction: 'query',
        valueField: 'value',
        displayField: 'text',
        applyTo: 'combo'
    });

    combo.on('select', function(comboBox){
        alert(comboBox.getValue() + '-' + comboBox.getRawValue());
    });
```

示例在04.form/07-06-01.html中。

4.7.7 使用本地数据实现省、市、县级联

下面我们来根据事件监听器来制作一个多ComboBox级联显示数据的示例。这是一个非常典型的示例，以前经常用select实现它，现在来看看用ComboBox实现的效果（见图4-32）。



图4-32 多ComboBox级联显示数据

实现图4-31中的效果的思路大致是这样的，我们先准备好省、市、县的数据（都是二维数组），级联显示的数据就是从它们之中获取的（见代码清单4-11）。

代码清单4-11 多ComboBox级联显示数据

```
var dataProvince = [
    ['河北','河北'],
    ['内蒙古','内蒙古']
];
var dataCityHebei = [
    ['唐山','唐山'],
    ['秦皇岛','秦皇岛'],
    ['承德','承德'],
    ['张家口','张家口']
];
var dataCityNeimenggu = [
    ['呼和浩特','呼和浩特'],
    ['包头','包头']
];
// 其中：遵化、迁安为县级市。
var dataCountyTangshan = [
    ['路南区','路南区'],
    ['路北区','路北区'],
    ['开平区','开平区'],
    ['古冶区','古冶区'],
    ['丰润区','丰润区'],
    ['丰南区','丰南区'],
    ['玉田','玉田'],
    ['遵化','遵化'],
    ['迁西','迁西'],
    ['迁安','迁安'],
    ['滦县','滦县'],
    ['滦南','滦南'],
    ['乐亭','乐亭'],
    ['唐海','唐海']
];
var dataCountyUnknow = [
    ['不知道','不知道']
];
```

然后我们需要3个数据转换器，分别对应省、市、县的Ext.data.SimpleData。原始数据经过它们的处理才可以提交给ComboBox，从而显示出我们的效果，如代码清单4-12所示。

代码清单4-12 对应的store

```
var storeProvince = new Ext.data.SimpleStore({
    fields: ['value', 'text'],
    data: dataProvince
});
var storeCity = new Ext.data.SimpleStore({
    fields: ['value', 'text'],
    data: []
});
var storeCounty = new Ext.data.SimpleStore({
    fields: ['value', 'text'],
    data: []
});
```

注意，上面的代码中只为storeProvince提供了预先定义好的二维数组dataProvince，其他两个store里使用的都是空数组[]。因为在没有选择省级地区之前，市级地区和县级地区是不应该有数据的。

现在可以制作3个ComboBox了。每个Combobox都配置成readOnly:true，这样可以避免用户输入无效信息。设置参数triggerAction:all避免ComboBox使用autoComplete的效果，让每次选择都可以看到所有的可选数据，具体实现如代码清单4-13所示。

代码清单4-13 设置ComboBox

```
var comboProvince = new Ext.form.ComboBox({
    store: storeProvince,
    emptyText: '请选择',
    mode: 'local',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text',
    readOnly: true
});
var comboCity = new Ext.form.ComboBox({
    store: storeCity,
    emptyText: '请选择',
    mode: 'local',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text',
    readOnly: true
});
var comboCounty = new Ext.form.ComboBox({
    store: storeCounty,
    emptyText: '请选择',
    mode: 'local',
    triggerAction: 'all',
    valueField: 'value',
    displayField: 'text',
    readOnly: true
});
comboProvince.applyTo('comboProvince');
comboCity.applyTo('comboCity');
comboCounty.applyTo('comboCounty');
```

在HTML里对应放上3个input，显示部分就基本完成了，如下面的代码所示。

```
<input id="comboProvince" type="text"/>
<input id="comboCity" type="text"/>
<input id="comboCounty" type="text"/>
```

现在我们来看看关键的地方，如何让3个ComboBox关联起来，选择上级时，下级显示的信息要发生相应的变化。这就要用到前面提到的on('select')执行事件监听了，如代码清单4-14所示。

代码清单4-14　利用on('select')执行事件监听

```
comboProvince.on('select', function(comboBox){
    var province = comboBox.getValue();
    if (province == '河北') {
        storeCity.loadData(dataCityHebei);
    } else if (province == '内蒙古') {
        storeCity.loadData(dataCityNeimenggu);
    }
});
comboCity.on('select', function(comboBox){
    var city = comboBox.getValue();
    if (city == '唐山') {
        storeCounty.loadData(dataCountyTangshan);
    } else {
        storeCounty.loadData(dataCountyUnknow);
    }
});
comboCounty.on('select', function(comboBox){
    alert(comboProvince.getValue() + '-' + comboCity.getValue() + '-' + comboCounty.
    getValue());
});
```

像上面的代码这样，给3个ComboBox都设置上on('select')的事件监听。在选择省时，先判断选择的是哪个省。如果我们选择的是河北，那么就要在市级ComboBox里显示河北省所管辖的城市的信息。

注意我们用到的storeCity.loadData(dataCityHebei);，这个函数可以改变store内部的数据。store的数据改变了，对应的ComboBox也就发生了变化。省和市的操作是一样的，最后在选择县时用alert弹出最终选择的信息。

上述过程就是我们使用ComboBox实现的省、市、县三级级联。

示例在04.form/07-07-01.html中。

4.7.8　使用后台数据实现省、市、县级联

在上面的示例中，省、市、县的信息都写在前台，现在我们来演示一下如何从后台获得数据并实现省、市、县三级级联。我们的后台采用JSP编写，脚本的名称为city.jsp，如代码清单4-15所示。

代码清单4-15　city.jsp

```
<%@ page contentType="text/html;charset=utf-8"%>
<%!
    String province = "[['河北','河北'],['内蒙古','内蒙古']]";
    String cityHebei = "[" +
        "['唐山','唐山']," +
        "['秦皇岛','秦皇岛']," +
        "['承德','承德']," +
        "['张家口','张家口']" +
    "]";
```

```
        String cityNeimenggu = "[" +
            "['呼和浩特','呼和浩特']," +
            "['包头','包头']" +
        "]";
        String countyTangshan = "[" +
            "['路南区','路南区']," +
            "['路北区','路北区']," +
            "['开平区','开平区']," +
            "['古冶区','古冶区']," +
            "['丰润区','丰润区']," +
            "['丰南区','丰南区']," +
            "['玉田','玉田']," +
            "['遵化','遵化']," +
            "['迁西','迁西']," +
            "['迁安','迁安']," +
            "['滦县','滦县']," +
            "['滦南','滦南']," +
            "['乐亭','乐亭']," +
            "['唐海','唐海']" +
        "]";
        String countyUnknow = "[['不知道','不知道']]";
%>
<%
request.setCharacterEncoding("UTF-8");
response.setCharacterEncoding("UTF-8");

String type = request.getParameter("type");
if ("province".equals(type)) {
    response.getWriter().print(province);
} else if ("city".equals(type)) {
    String province = java.net.URLDecoder.decode(request.getParameter("id"));
    System.out.println(province);
    if ("河北".equals(province)) {
        response.getWriter().print(cityHebei);
    } else if ("内蒙古".equals(province)) {
        response.getWriter().print(cityNeimenggu);
    }
} else if ("county".equals(type)) {
    String city = request.getParameter("id");
    if ("唐山".equals(city)) {
        response.getWriter().print(countyTangshan);
    } else {
        response.getWriter().print(countyUnknow);
    }

}
%>
```

需要注意的是，因为Ajax默认使用UTF-8编码传输数据，所以需要把示例中的文件都转换成UTF-8编码格式，否则就会出现乱码。还要对`request`和`response`进行处理，统一使用UTF-8编码。建议所有项目文件都统一使用UTF-8编码，这样会有更好的扩展性。

仔细查看city.jsp中的内容，文件开头的`<%!%>`部分定义了我们需要返回的省、市、县的数据。这里使用了JSON格式的字符串，回传到页面后，EXT会把它们解析成对应的JSON数组。

主程序体很简单，先获得type的值，进而判断后面要读取省、市、县哪一部分的信息，根据type分别进入各自的流程。

后台准备完毕，现在对前台进行修改。先将Ext.data.SimpleStore改成Ext.data.Store，使用HttpProxy指向我们准备好的city.jsp，然后控制store在选择不同ComboBox时向后台传递不同的type参数，如代码清单4-16所示。

代码清单4-16　远程store

```
var storeProvince = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url:'city.jsp?type=province'}),
    reader: new Ext.data.ArrayReader({},[
        {name:'value'},
        {name:'text'}
    ])
});
var storeCity = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url:'city.jsp?type=city'}),
    reader: new Ext.data.ArrayReader({},[
        {name:'value'},
        {name:'text'}
    ])
});
var storeCounty = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url:'city.jsp?type=county'}),
    reader: new Ext.data.ArrayReader({},[
        {name:'value'},
        {name:'text'}
    ])
});
```

从上例中可以看出，除了HttpProxy中url的最后type部分不同外，其他代码都是一样的。

创建ComboBox的其他部分完全一样，mode依然使用local方式，因为要控制读取数据的时机，所以手动运行store.load()更符合我们的需要。

对on('select')事件监听器进行了一些修改，控制对应的store向后台发送请求，如代码清单4-17所示。

代码清单4-17　事件监听（远程）

```
storeProvince.load();
comboProvince.on('select', function(comboBox){
    var value = comboBox.getValue();
    storeCity.load({params:{id:value}});
});
comboCity.on('select', function(comboBox){
    var value = comboBox.getValue();
    storeCounty.load({params:{id:value}});
});
comboCounty.on('select', function(comboBox){
    alert(comboProvince.getValue() + '-' + comboCity.getValue() + '-' + comboCounty.
    getValue());
});
```

从上述代码可以看出，我们先执行storeProvince.load();，将省级数据初始化。然后判断用户选择了省级信息的哪一项，取得数据以后再调用市级ComboBox的load()函数去后台读取数据。这时我们选择的省级信息会被当作参数传递给后台，后台根据传递过来的id值判断究竟要返回省、市、县哪一级的信息数据。

注意　为了统一编码，07-08-01.html和city.jsp都另存为UTF-8编码，它们都放在本书示例的04.form目录下。

4.8 复选框和单选框

从组件继承结构图中可以看到，复选框（Checkbox）和单选框（Radio）是与文本框（TextField）完全不同的分支。之前讨论到的验证和布局功能有很大的区别，所以要对它们进行单独的讨论。

4.8.1 复选框

要在表单里加上复选框就需要用Ext.form.Checkbox，可惜的是我们无法控制复选框的样式，与其他表单控件相比，它显得十分难看（见图4-33）。

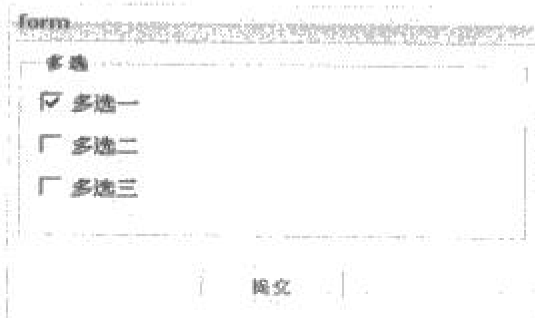


图4-33　复选框

从图4-33中可以看出，我们用一个Fieldset把三个复选框放在了一起。但是，你有没有注意到，复选框对应的标签都位于选择框的右边。这并不是我们控制了标签的显示位置，而是因为使用了boxLabel的原因。boxLabel是复选框和单选框两个控件独有的，支持在控件右侧显示标签。它与labelField只是位置不同，具体应用中可以随意选择，如下面的代码所示。

```
items: [{
    xtype: 'fieldset',
    title: '多选',
    autoHeight:true,
    defaultType: 'checkbox',
    hideLabels: true,
    items: [
        {boxLabel: '多选一', checked: true},
        {boxLabel: '多选二'},
        {boxLabel: '多选三'}
    ]
}],
```

注意　为了避免出现前后两个标签，还是把hideLabels设置为true比较好，这样左侧的fieldLabel就不会显示出来了。

现在我们可以单击“提交”按钮，但是后台无法区分这3个复选框，因为value的默认值都是on，提交的数据就变成这样：

```
checkbox=on&checkbox=on&checkbox=on
```

怎么区分这3个复选框呢？我们需要使用inputValue来指定这3个复选框的值，如下面的代码所示。

```
items: [{
    xtype: 'fieldset',
    title: '多选',
    autoHeight:true,
    defaultType: 'checkbox',
    hideLabels: true,
    items: [
        {boxLabel: '多选一', inputValue: '1', checked: true},
        {boxLabel: '多选二', inputValue: '2'},
        {boxLabel: '多选三', inputValue: '3'}
    ]
}],
```

再次提交就会得到这样的结果：

```
checkbox=1&checkbox=2&checkbox=3
```

怎么样才能让这几个Checkbox一开始就是选中状态呢？我们可以使用checked: true参数，上面示例中的第一个Checkbox就演示了这种效果。

示例在04.form/08-01-01.html中。

4.8.2　单选框

EXT中的单选框（Radio）是继承自复选框的，所以复选框中的所有功能都能在单选框中使用。唯一的区别是，当你希望制作一组单选框并限制每次只能选一个时，就要进行一些有别于复选框的配置了，如下面的代码所示。

```
items: [{
    xtype: 'fieldset',
    title: '单选',
    autoHeight:true,
    defaultType: 'radio',
    hideLabels: true,
    items: [
        {boxLabel: '单选一', name: 'radio', inputValue: '1', checked: true},
        {boxLabel: '单选二', name: 'radio', inputValue: '2'},
        {boxLabel: '单选三', name: 'radio', inputValue: '3'}
    ]
}],
```

注意3个单选框的name参数，具有相同名称的单选框会放在同一组，这样就可以保证同一组只有一个单选框被选中，如图4-34所示。

因为名称都是一样的，inputValue就显得尤为重要了，否则我们无法判断用户选择了哪个

单选框。

除了复选框所拥有的功能之外，单选框还有一个自己独有的函数getGroupValue()。这个函数可以获得某个分组中被选中的单选框的值，我们再也不用手工查找同一分组中哪个单选框被选中了。在示例中，我们专门设置了一个按钮来演示getGroupValue()效果，你也可以试试看。

示例在04.form/08-02-01.html中。



图4-34 单选框

4.9 文件上传

如何在EXT中使用文件上传组件呢？方法其实很简单，为Ext.form.Field设置inputType:'file'即可，但我们无法修改它的显示样式，如图4-35所示。

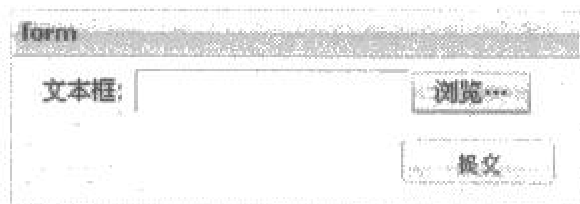


图4-35 文件上传

我们现在需要了解如何才能让这个表单实现文件上传功能。

首先，为表单添加fileUpload: true参数，如下面的代码所示。

```
var form = new Ext.form.FormPanel({
    labelAlign: 'right',
    title: 'form',
    labelWidth: 50,
    frame:true,
    fileUpload: true,
    url: '09_01.jsp',
    width: 280,
```

其次，给表单添加一个field，并为它设置inputType: 'file'，如下面的代码所示。

```
items: [{
    xtype: 'textfield',
    fieldLabel: '文本框',
    name: 'file',
    inputType: 'file'
}],
```

现在单击“提交”按钮就可以上传文件了。EXT中使用Ajax实现文件上传，不需要刷新整个页面。而在后台处理文件上传的09_01.jsp中，我们使用了commons-fileupload组件处理客户上传的文件，运行示例程序之前，先要把commons-io和commons-fileupload放到lib下，这样才可以使用。

示例在04.form/09_01.html中。

当然，这里使用的inputType:'file'也只是HTML中的原生file组件，连外观也没有任何变化。在EXT的官方网站上可以找到一个上传的扩展件Ext.ux.UploadDialog。关于如何使用扩展件的问题已经超出了本书的范围，建议大家自己尝试一下。

4.10 自动把数据填充到表单中

添加数据与修改数据的操作是相辅相成的，很少有只允许添加而不允修改的情况。如果要进行修改，还是使用原来的表单，我们需要做的就是在表单显示时为每个控件赋予对应的数据。我们知道 Ext.form.Field都有setValue()函数，可以设置表单中对应控件的数据。但是，把这些控件逐个取出来，然后再逐个赋值，还有很多数据要进行类型转换，实在是很麻烦。

我们有一个包含了如下控件的表单，如图4-36所示。

如果只需要单击"读取"按钮就可以把对应的数据自动填充到每个控件中（见图4-37），并且附带数据类型转换，那就更方便了。

图4-36 准备填入数据的表单

图4-37 填入数据后的表单

我们当然不会使用setValue()为每个控件填充数据。为了让复杂的工作变得简单，我们使用Ext.data.JsonReader来负责数据的读取和转换操作。

后台传过来的数据是只有一个元素的JSON数组，如下面的代码所示。

```
[{
    text: 'textField',
    number: 12.34,
    date: '2008-01-01T00:00:00',
    combo: 1
}]
```

这里提供了字符串、数字、日期等类型的数据，表单中需要配置对应的reader，如下面的代码所示。

```
var reader = new Ext.data.JsonReader({},[
    {name: 'text', type: 'string'},
    {name: 'number', type: 'float'},
    {name: 'date', type: 'date', dateFormat: 'Y-m-dTH:i:s'},
    {name: 'combo', type: 'int'}
]);
```

现在我们将设置好的reader放到表单中，后台返回的JSON会在这里被JsonReader转换成对应的数据类型，供表单使用，如下面的代码所示。

```
var form = new Ext.form.FormPanel({
    labelAlign: 'right',
    title: 'form',
    labelWidth: 50,
    frame:true,
    url: '09_01.jsp',
    width: 280,
    reader: reader,

items: [{
            xtype: 'textfield',
            fieldLabel: '文本',
            name: 'text'
        },{
            xtype: 'numberfield',
            fieldLabel: '数字',
            name: 'number'
        },{
            xtype: 'datefield',
            fieldLabel: '日期',
            name: 'date'
        },{
            xtype: 'combo',
            fieldLabel: '下拉',
            name: 'combo',
            store: new Ext.data.SimpleStore({
                fields: ['value', 'text'],
                data : [
                    [1, 'text1'],
                    [2, 'text2'],
                    [3, 'text3']
                ]
            }),
            triggerAction: 'all',
            valueField: 'value',
            displayField: 'text'
}]
```

当调用form.load()函数时，表单会使用Ajax去后台读取需要的数据。如果调用load()时没有使用任何参数，load()函数就会使用表单中对应的url参数。不过表单中设置的url一般都是提交数据的网址，为了不将提交和读取这两个操作混在一起，我们建议另外定义一个专门用来读取数据的url，如下面的代码所示。

```
{
    text: '读取',
    handler: function() {
        form.getForm().load({url:'10-01.txt'});
    }
}
```

现在我们为load()传递一个url参数，指定读取数据的网址。这个网址返回的信息就是上面提到的用于向表单填充数据的JSON字符串，这样就实现了自动为表单中的各个组件填充数据的功能。

除了load()函数外，表单还提供了一个loadRecord()函数。因为在EXT的组件体系中，很多地方都会用到Ext.data.Store和Ext.data.Record。例如，最常见的是表格和ComboBox，通过loadRecord()，我们可以把表格中的一条数据加载到表单中，从而进行修改。

示例在04.form/10-01.html中。

提示　从EXT 3.0开始，表单可以选择使用Ajax或是Ext Direct方式来进行数据提交和数据读取，有关Ext Direct的介绍可以在第14章中找到。

4.11　小结

本章主要介绍了EXT中的表单以及表单中使用的输入组件，通过组件继承图我们可以清楚地了解到它们之间的相互关系。我们以Ext.form.TextField为基础，介绍了常用组件的功能以及数据校验方式，并特别介绍了Ext.form.ComboBox、Ext.form.Checkbox、Ext.form.Radio以及上传组件的使用方法，同时还讲解了使用ComboBox如何实现三级联动。

本章还讲解了表单的提交、数据校验、表单布局，最后介绍了如何将外部数据填充到表单内的组件中。

第5章

树 形 结 构

5

本章内容

- TreePanel的基本使用
- 树的事件
- 右键菜单
- 修改节点的默认图标
- 从节点弹出对话框
- 节点提示信息
- 为节点设置超链接
- 直接修改树节点名称
- 树形的拖放
- 树形过滤器TreeFilter
- 利用TreeSorter对树进行排序
- 树形节点视图——Ext.tree.TreeNodeUI
- 表格与树形的结合——Ext.ux.tree.ColumnTree

5.1 TreePanel 的基本使用

在应用程序中，我们经常需要显示或处理树状结构的对象信息，比如部门信息和地区信息（省、市、县）等。树是一种非常典型的数据结构，这些信息都可以用树表示。

对传统的HTML页面来说，完全依靠手动编码来实现树是比较困难的，因为需要写很多的JavaScript代码。对基于Ajax异步加载的树来说更是如此，不但涉及Ajax数据异步加载，还需要考虑跨浏览器支持，处理起来非常麻烦。EXT中提供了现成的树控件，通过这些控件，可以在B/S应用中快速开发出包含树形信息结构的应用。

5.1.1 创建一棵树

树控件由Ext.tree.TreePanel类定义，控件的名称为Treepanel，TreePanel类继承自Panel面板。在EXT中使用树控件其实非常简单，我们先来看下面的代码。

```
var tree = new Ext.tree.TreePanel({
    el: 'tree'
});
```

这里的参数tree表示渲染的DOM的id。HTML中有<div id="tree"></div>相对应，最后这棵树就出现在这个div的位置上。

现在，我们获得了树形面板。既然是树，就必须有一个根，有了根才能在上面生长枝节，最后成为一棵完整的树，所以根是必须的。我们通过下面的代码看看根是如何生长的。

```
var root = new Ext.tree.TreeNode({text:'我是根'});
tree.setRootNode(root);
tree.render();
```

上面代码的第1行就是定义一棵树的根。我们用setRootNode()方法把根root放到树形里。然后对树形进行渲染，让它出现在id="tree"的位置，这个id是在前面代码里指定的，如果有疑问，请查阅前面的内容。

这样我们就创建了一棵只有根的树，如图5-1所示。

我是根

图5-1　只有根的树

当然，它现在怎么看都不像是树，因为其实只创建了一棵树的根。下面就可以在这个根上面开枝散叶了。

注意　虽然只有一个根，不过也算是棵树，示例在文件05.tree/01-01-01.html中。

5.1.2　为树生枝展叶

上面已经创建了一棵只有根的树，下面我们就为树添加枝和叶，让它看上去更像一棵树，如代码清单5-1所示。

代码清单5-1　为树添加枝和叶

```
var root = new Ext.tree.TreeNode({text:'我是根'});
var node1 = new Ext.tree.TreeNode({text:'我是根的第一个枝'});
var node2 = new Ext.tree.TreeNode({text:'我是根的第一个枝的第一个叶子'});
var node3 = new Ext.tree.TreeNode({text:'我是根的第一个叶子'});
node1.appendChild(node2);
root.appendChild(node1);
root.appendChild(node3);
```

我们创建了3个TreeNode，然后把node2插到node1上，node1、node2一起插到root上。这样node2成了叶，node1是枝，长在根root上。node3直接生长在根上，而下面又没有再长出任何东西，便也被称作是叶，现在就更像是一棵树了（见图5-2）。

我是根

图5-2　添加枝叶后的树（未展开）

现在，这棵树看上去仍像是一个根，但图标却发生了变化。图标前多了个加号，现在的根可以展开了，如图5-3所示。

树展开后，就可以看到它的枝和叶了。不过，每次都要点击根或枝前面的加号才能展开整棵树。这样过于麻烦，我们可以让树加载后就立刻展开，如下面的代码所示。

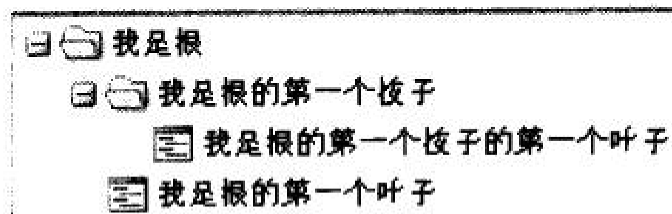


图5-3 添加枝叶后的树（已展开）

```
root.expand(true, true);
```

只用一行简单的代码就实现了我们想要的功能。上面代码里的第一个参数表示是否递归展开所有子节点，如果为false，就只展开第一级子节点，下面的子节点仍然是折叠状态。第二个参数表示是否要动画效果，如果为true，可以明显看出这些节点是逐渐展开的。当然，它们也可以直接展开。

为了方便，在我们的示例中是直接展开的。

示例在05.tree/01-02-01.html中。

5.1.3 树形的配置

下面我们修改一下TreePanel的定义，原来的id要放到{}中，对应的名字是el，修改后的代码如下所示。

```
var tree = new Ext.tree.TreePanel({
    renderTo: 'tree',
    root: new Ext.tree.TreeNode({text:'我是根'})
});
```

但是，这样改完之后还是没有内容显示。用Firebug查看DOM，height竟然为0，当然看不到任何内容了。因为树不能自动计算自身的高度，我们只好给它设置一个初始高度。在HTML里设置这个高度为300（px），这样就可以显示出内容了。

如下面的代码所示：

```
<div id="tree" style="height:300px;"></div>
```

如果不想设置div的CSS高度，我们也可以加入autoHeight:true，让treePanel自己计算显示的高度。我们还可以看到鼠标移到树节点上时的突出显示，如图5-4所示。

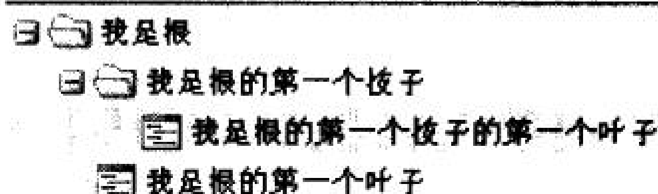


图5-4 突出显示

看完这些示例，我们应该对树有些了解了。虽然这里只有TreeNode，却能表示枝杈或者叶子。原理很简单，如果这个TreeNode下有其他节点，它就是一个枝杈；如果没有，它就是一个叶子，从它前面的图标就很容易看出来。根其实就是一个没有上级节点的枝杈。实际上，它们都

是TreeNode，只是属性不同而已。

该示例在05.tree文件夹下，分别是01-01-01.html和01-02-01.html。

5.1.4　使用 TreeLoader 获得数据

像上面那样获取数据不仅麻烦，而且还容易出错。Ext.tree.TreeLoader可以利用从后台获取的数据为我们组装出一棵树来，我们只需要提供数据，让TreeLoader完成数据转换和装配节点的操作。

这里又需要用到JSON和Ajax了。重申一下前面提到过的问题：一旦涉及Ajax，就需要配合服务器，Ajax是无法从本地文件系统直接取得数据的。

首先，为TreePanel配置一个TreeLoader，如下面的代码所示。

```
var tree = new Ext.tree.TreePanel({
    el: 'tree',
    loader: new Ext.tree.TreeLoader({dataUrl: '01-04-01.txt'})
});
```

这里的TreeLoader仅包含一个参数dataUrl: '03-03.txt'，这个dataUrl表示在树完成渲染后从何处读取数据。为了演示方便，我们写了一个文本文件来为树形提供JSON数据，打开03-03.txt可以看到里边的内容。

如下面的代码所示：

```
[
    {text:'not leaf'},
    {text:'is leaf',leaf:true}
]
```

这是一个包含了两个节点的数组，你可能会发现附加属性leaf:true，这个属性的作用将会在下面讲到。

如果现在刷新页面，你依然只能看到根，没有像你期待的那样从03-03.txt读取数据并显示到页面上。因为我们使用的TreeNode并不支持Ajax，需要把根节点换成AsyncTreeNode，这样才可以实现我们想要的异步加载效果，如下面的代码所示。

```
var root = new Ext.tree.AsyncTreeNode({text:'我是根'});
```

结果如图5-5所示，叶子无限展开。

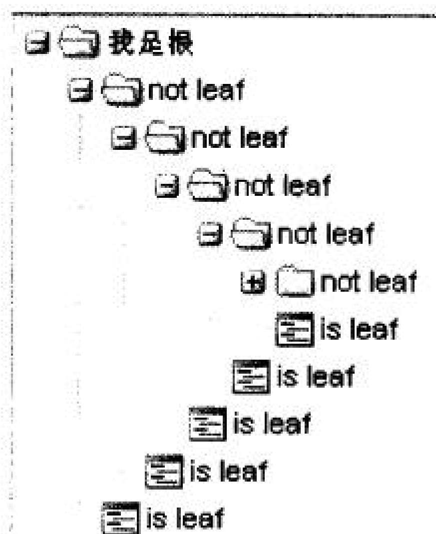


图5-5　叶子无限展开

图5-5的效果还是和我们预想的不一样，树的节点竟然是无限循环展开的。我们把root.expand(true, true)改成root.expand()，避免节点无限展开下去，现在分析一下出现这种结果的原因。

取消了递归展开之后，树只展开根节点的第一层节点（见图5-6）。从图5-6中可以看出，我们得到的确实是与03-03.txt文件里相对应的两个节点。不过，这两个节点却有些不同，not leaf节点的图标显然是枝杈的图标，如果点击前面的加号，它还会

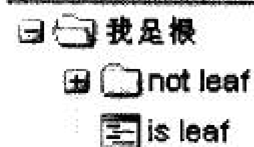


图5-6　取消递归展开后的效果

像图5-5那样向下展开。

原因就在于AsyncTreeNode，它会继承树形TreeLoader中的dataUrl。当展开节点执行对应节点的expand()方法时，它会通过Ajax访问dataUrl指定的地址并获取数据，而且还会把自己的id作为参数传递给dataUrl指定的地址。通过Firebug可以看到Ajax请求的整个过程，而我们的后台应该通过节点id计算出应该返回的数据，TreeLoader会根据获得的相应数据为树形装配节点，并显示到页面上。

关键就是这里，因为我们使用03-03.txt提供的数据不会判断当前节点的id，所以每次返回的数据都是一样的，这样树会无限循环下去。

大家可能会问："为什么只有第一个节点会无限循环下去，而第二个节点却没有那个小加号？"这是因为第二个节点不是AsyncTreeNode。TreeLoader在生成节点时会判断数据里的leaf属性，如果是leaf:true，就会生成TreeNode，而不是AsyncTreeNode。TreeNode不会自动去用Ajax取值，自然就不会无限循环展开了。

现实中，异步读取属性的节点是非常好的一种方法，因为可能要保存成千上万条节点记录。如果一次性全部装载，读取和渲染的速度都会很慢。如果使用异步读取的方式，那么只有在点击某一节点后才会去获取子节点属性并进行渲染，极大提高了用户体验。而且，EXT的树本身有缓存机制，打开一次后，再次点击时不会去重复读取数据了，这也提升了响应速度。

示例在05.tree/01-04-01.html中。

下面我们再写一个从JSON获取数据的示例，对前面所讲的知识进行巩固和实践，以此加深理解。这次的JSON文件稍微复杂些，对应的保存着JSON数据的文件是01-04-02.txt。

01-04-02.txt的内容如下所示。

```
[
    {text:'01',children:[
        {text:'01-01',leaf:true},
        {text:'01-02',children:[
            {text:'01-02-01',leaf:true},
            {text:'01-02-02',leaf:true}
        ]},
        {text:'01-03',leaf:true}
    ]},
    {text:'02',leaf:true}
]
```

运行结果如图5-7所示。

这也可以看作是在数据不多的情况下一次加载所有数据的途径。只要确保在所有叶子节点上都加上leaf:true的属性，就不会出现循环展开的问题。

示例在05.tree/01-04-02.html中。

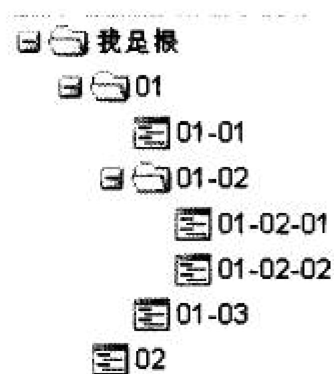


图5-7 异步加载树完全展开

5.1.5 读取本地 JSON 数据

读取本地JSON其实是使用TreeLoader的另一种方式。因为有时树形的数据并不多，为了获取如此少量的数据而使用Ajax访问后台实在不划算。可是，如果退回到每个节点都使用new来生

成，又实在太麻烦了。那么，能不能让TreeLoader读取本地JavaScript中的JSON数据，然后生成需要的树形节点呢？

答案当然是肯定的。首先，为TreePanel设置一个参数为空的TreeLoader，如下面代码所示。

```
var tree = new Ext.tree.TreePanel({
    el: 'tree',
    loader: new Ext.tree.TreeLoader()
});
```

然后，设置一个根节点，并为这个根节点设置children属性，如下面代码所示。

```
var root = new Ext.tree.AsyncTreeNode({
    text:'我是根',
    children: [
        {text: 'Leaf No. 1',leaf: true},
        {text: 'Leaf No. 2',leaf: true}
    ]
});
```

这里有以下3点需要注意。

- 必须设置TreeLoader，否则根节点会一直处在读取状态，无法获得children中定义的子节点。
- 根节点必须是AsyncTreeNode，如果是TreeNode，也无法生成子节点。
- 子节点中的叶子节点必须都加上leaf: true，否则也会一直显示读取状态。

示例在05.tree/01-05-01.html中。

5.1.6 与 Struts 2 进行集成

接下来将讨论如何使用TreeLoader与Struts 2和Spring MVC进行集成。

问题来源于Struts 2提供的JsonPlugin与Spring MVC提供的JsonView组件的应用。事件的起因是这样的，TreeLoader只能处理JSON数组，也就是后台返回的数据必须包含在[]中。然而，JsonPlugin和JsonView两个组件都只能返回JSON对象，即返回的数据都包含在{}中，于是矛盾就出现了。

一般说来，我们不应该去修改后台，直接使用JsonPlugin和JsonView会更方便，也更便于以后的版本升级。这样一来，就需要修改TreeLoader了。

下面给出一种修改TreeLoader的方法，只修改TreeLoader中的一个函数，使其可以支持对象。但是，这个对象的某个值必须包含数组。

现在假设后台生成的JSON对象如下面的代码所示：

```
{key:[
    {text:'01'},
    {text:'02',leaf:true}
]}
```

key对应的值就是树形需要的节点数据。接下来我们修改一下TreeLoader函数，如代码清单5-2所示。

代码清单5-2 修改TreeLoader

```
var loader = new Ext.tree.TreeLoader({dataUrl: '01-06-01.txt'});
loader.processResponse = function(response, node, callback){
    var json = response.responseText;
    try {
        var json = eval("("+json+")");
        node.beginUpdate();
        // 从json中取得json数组
        var o = json["key"];

        for(var i = 0, len = o.length; i < len; i++){
            var n = this.createNode(o[i]);
            if(n){
                node.appendChild(n);
            }
        }
        node.endUpdate();
        if(typeof callback == "function"){
            callback(this, node);
        }
    }catch(e){
        this.handleFailure(response);
    }
};
```

实际上我们只添加了一行代码`var o = json["key"]`，把key对应的数组值取出来，其他的代码都没有改变，还是按原来的方式处理获得的数据。

如果只有一处或两处用到树形，那么这样修改也就足够了，记得每次都要把key改成实际返回的值。如果还有更多地方需要进行JSON数据转换，建议对TreeLoader进行扩展，以后可以直接引用扩展后的TreeLoader。

示例在05.tree/01-06-01.html中。

5.1.7 使用 JSP 提供后台数据

我们在后台使用JSP，下面来看看如何判断目前展开的节点并进行处理。既然有了后台，就需要用到服务器，我们这里使用的是Tomcat。关于Tomcat的安装和使用，请参考网站www.family168.com上的JSP教程，这里就不再赘述。

前台的HTML和JavaScript代码使用之前的示例，但是要把TreeLoader中的dataUrl改成我们现在用到的tree.jsp，如下面的代码所示。

```
var tree = new Ext.tree.TreePanel({
    el: 'tree',
    loader: new Ext.tree.TreeLoader({dataUrl: '01_07_01.jsp'})
});
```

另外，还要给root节点设置一个id，这样后台才能知道应该在何时返回根节点对应的子节点数据，如下面的代码所示。

```
var root = new Ext.tree.AsyncTreeNode({
```

```
    id: '0',
    text: '我是根'
});
```

将root的id设置为'0'。注意，树形中节点的id不要重复。在EXT中，如果出现重复的id，就会出现问题。随后，后台会根据id来判断究竟是哪个节点正在展开，从而返回对应的数据。

前台准备就绪，下面看一下后台01_07_01.jsp的脚本，如代码清单5-3所示。

代码清单5-3　后台JSP代码

```
<%@ page contentType="text/html;charset=utf-8"%>
<%
    request.setCharacterEncoding("UTF-8");
    response.setCharacterEncoding("UTF-8");

    // 获得node参数，对应的是正在展开的节点id
    String node = request.getParameter("node");
    System.out.println(node);

    String json = "";
    if ("0".equals(node)) {
        json += "[{id:1,text:'节点阿一'},{id:2,text:'节点阿二'}]";
    } else if ("1".equals(node)) {
        json += "[{id:11,text:'节点阿一一'},{id:12,text:'节点阿一二'}]";
    } else if ("2".equals(node)) {
        json += "[{id:21,text:'节点阿二一'},{id:22,text:'节点阿二二'}]";
    } else if ("11".equals(node)) {
        json += "[{id:111,text:'节点阿一一一'},{id:112,text:'节点阿一一二'}]";
    }

    response.getWriter().print(json);
%>
```

要强调一点，因为Ajax默认使用UTF-8编码格式，所以我们的JSP也要使用UTF-8编码发送数据。

其实，树形异步读取的关键是node参数。当某个节点展开时，TreeLoader会根据设置的dataUrl地址去后台读取数据。而当发送请求时，TreeLoader会把这个节点的id作为参数一起发送到后台去。对后台来说，只要获得node参数，就知道是哪个节点正在执行展开。

剩下的就简单了，我们根据节点的id返回对应的JSON数据。返回的响应中必须包含每个节点的id和text，id是节点的唯一标识，text代表节点的名称。其他属性我们暂时还没有接触到，稍后再对它们进行讨论。前台展示树的代码如下所示：

```
var tree = new Ext.tree.TreePanel({
    el: 'tree',
    loader: new Ext.tree.TreeLoader({dataUrl: '01_07_01.jsp'})
});

var root = new Ext.tree.AsyncTreeNode({
    id: '0',
    text:'我是根'
```

```
});

tree.setRootNode(root);
tree.render();

root.expand(false, false);
```

这样就完成了树从后台读取数据并展示的效果。从实际操作中可以看到，展开节点时会提示正在读取数据，读取成功后就会显示节点下面的所有子节点，如图5-8所示。

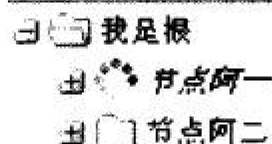


图5-8 异步加载树形节点

> **注意** 这时不能使用root.expand(true)进行递归展开。如果使用了expand(true)会导致树形节点不断向后台发送请求，直到所有节点都展开为止，无法实现异步的效果。

实际操作中，不知道你是否注意到一个奇怪的现象，如图5-9所示。

如果我们继续展开节点，图5-9会变成图5-10的样子。



图5-9 继续异步加载的结果

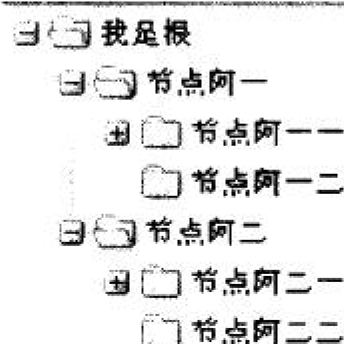


图5-10 异步加载没有返回子节点

节点并没有展开，但是它们前面的加号图标消失了。正如前面所说的，AsyncTreeNode会根据后台返回的数据生成不同的节点。如果返回的子节点数据包含leaf:true属性，就会生成TreeNode节点，并标记成叶子节点。但是，TreeNode不能再自动展开了。

现在的情况是，这几个节点本来是AsyncTreeNode，但当它们去后台读取数据时，后台没有返回任何子节点数据。这时候AsyncTreeNode才意识到自己犯了错误，自己很可能已经是叶子了，下面当然不会再有任何数据了。所以它急中生智，摇身一变，把自己变成了叶子模样，直接隐藏了表示展开的加号图标，这样它就变成了一个真正的叶子了。

虽然AsyncTreeNode能自己判断并生成不同的结果，但这个变化过程却降低了用户体验，因为用户并不知道其中原因。实际上，访问后台数据也是要花时间的，即使局域网速度很快，也会消耗服务器的带宽。无论如何，让节点自己判断自己是不是叶子，实在是个很差劲的方法。

那该怎么办呢？我们的解决方案还是配置leaf:true。修改过程很简单，我们在后台返回数据时进行判断。如果是叶子，就设置为leaf:true，这样前台显示就不会有问题了。修改后的01_07_02.jsp文件如代码清单5-4所示。

代码清单5-4　设置leaf:true

```
<%@ page contentType="text/html;charset=utf-8"%>
<%
    request.setCharacterEncoding("UTF-8");
    response.setCharacterEncoding("UTF-8");

    // 获得node参数，对应的是正在展开的节点id
    String node = request.getParameter("node");
    System.out.println(node);

    String json = "";
    if ("0".equals(node)) {
        json += "[{id:1,text:'节点阿一'},{id:2,text:'节点阿二'}]";
    } else if ("1".equals(node)) {
        json += "[{id:11,text:'节点阿一一',leaf:false},{id:12,text:'节点阿一二',leaf:true}]";
    } else if ("2".equals(node)) {
        json += "[{id:21,text:'节点阿二一',leaf:true},{id:22,text:'节点阿二二',leaf:true}]";
    } else if ("11".equals(node)) {
        json += "[{id:111,text:'节点阿一一一',leaf:true},{id:112,text:'节点阿一一二',leaf:true}]";
    }

    response.getWriter().print(json);
%>
```

前台的展示代码只需将TreeLoader里的dataUrl改成01_07_02.jsp即可，示例在05.tree/01-07-01.html中。

5.2　树的事件

EXT很奇特的一点是，它的事件模型会告诉你很多你想知道的事情，比如哪个节点展开了，哪个节点折叠了，树节点被单击还是双击了。而这一切在EXT的事件提醒机制下都会变得很简单，只需要使用on为树形注册事件监听器，如代码清单5-5所示。

代码清单5-5　监听树形事件

```
tree.on("expandnode", function(node){
    console.log(node + "展开了");
});
tree.on("collapsenode", function(node){
    console.log(node + "折叠了");
});
tree.on("click", function(node){
    console.log("你单击了" + node);
});
tree.on("dblclick", function(node){
    console.log("你双击了" + node);
});
```

接下来我们在树形上执行单击操作，展开或折叠其中的节点，如图5-11所示。

这里重点介绍一下图5-11中的Ext Debug Console，只有引入了ext-all-debug.js才会出现图中左

下角的效果。因为如果使用`alert()`进行测试，总是无法触发双击事件，所以只好借助它来显示效果了。为了使用`console.log()`方法，我们特意将以前的ext-all.js换成了ext-all-debug.js。EXT 2.*x*版本使用`Ext.log`来输出日志的，这样一来，即使我们不使用Firefox和Firebug也能清楚看到这些日志了。EXT 3.0估计是为了节省性能，直接在firebug的控制台输出日志，如图5-12所示。

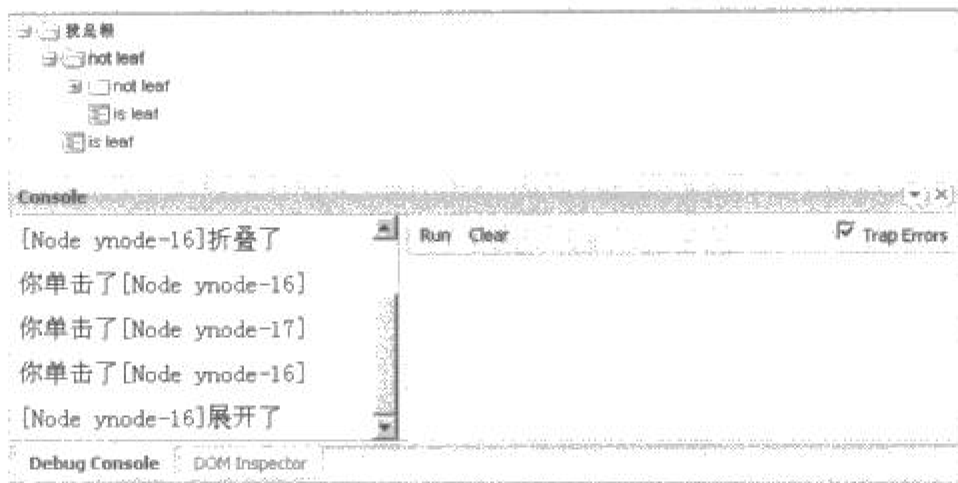


图5-11　Ext 2.*x*的debug监听树形事件

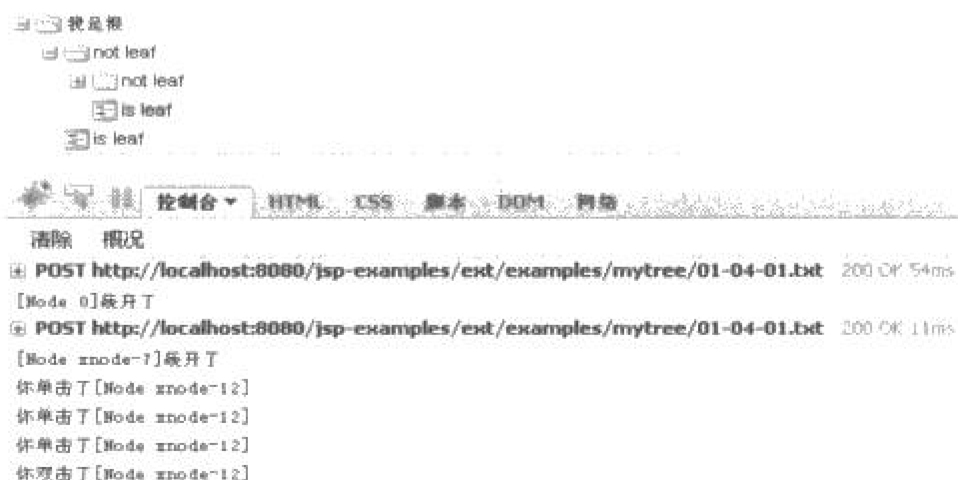


图5-12　Ext 3.0的debug监听树形事件

其实`on("eventName")`的用法非常简单，它只是把一个函数绑定到一个事件上，而我们只要知道这个事件的名字就可以了。当这个事件发生时，EXT会找到所有绑定好的函数，然后逐个执行。如此简单的原理却实现了如此绚丽的功能，我们只需要到API文档中查找究竟需要处理哪些事件，然后使用on进行绑定就可以了。

示例在05.tree/02-01.html中。

5.3　右键菜单

树形弹出的右键菜单效果如图5-13所示。

效果似乎比较简单，但实现过程却比较复杂。当然，它也依托了EXT的事件模型。我们要注册一个名为`contextmenu`的事件，当这个事件发生时，弹出自己定制的菜单。

下面我们来讨论一下实现的具体步骤。

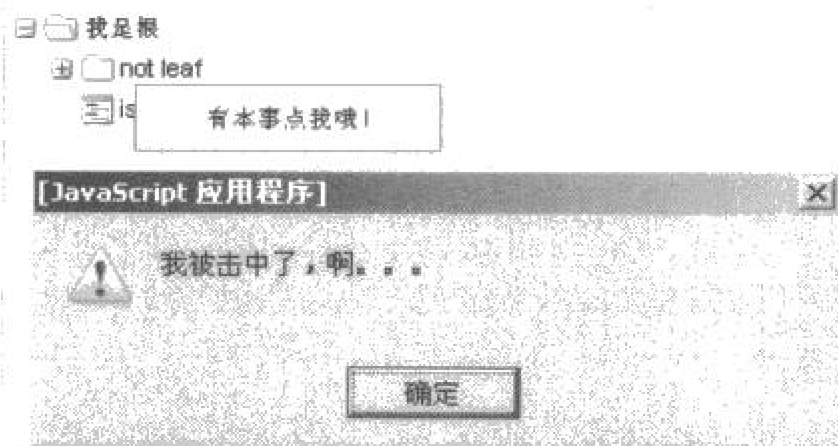


图5-13　树形弹出的右键菜单效果

(1) 制作自定义菜单，如下面的代码所示。

```
var contextmenu = new Ext.menu.Menu({
  id: 'theContextMenu',
    items: [{
        text: '有本事点我哦！',
        handler: function(){
            alert('我被击中了，啊。。。');
        }
    }]
});
```

像这样创建一个菜单，菜单里仅有一项，我们为这个菜单设置好单击时的处理函数，这些都是为右键事件做的准备工作。

(2) 绑定contextmenu事件，如下面的代码所示。

```
tree.on("contextmenu", function(node, e){
    e.preventDefault();
    node.select();
    contextmenu.showAt(e.getXY());
});
```

这个事件会向绑定的监听函数传递两个参数：当前点中的节点和事件对象。实际上，其他事件也会向绑定的监听函数传递这两个参数，我们通常只使用第一个参数。

首先，调用e.preventDefault()。这个函数的作用是防止浏览器弹出它默认的功能菜单，否则会一次弹出两个菜单：我们自定义的右键菜单和系统功能菜单，这显然乱套了。

然后，我们调用node.select()选中当前节点。因为右键单击事件虽然发生了，但当前节点可能没有处于选中状态，我们让它变成选中状态，可以避免以后会出现的问题。

showAt(e.getXY())方法通过右键事件e获得当前鼠标的坐标，右键菜单应该显示在这个坐标下，看上去就像是从节点上弹出来的。

至此，右键菜单的监听都已经完成，我们在弹出的右键菜单外任意点击一下鼠标，菜单应该会隐藏起来。但也存在意外情况，有时候菜单还没有隐藏起来我们就要执行其他操作，比如展开树形中的一个节点。此时如果菜单无法消失就会很碍眼，这时我们需要手工调用contextmenu.hide()函数让右键菜单消失。把该功能代码添加到实现展开、折叠操作功能的代码之前即可。

示例在05.tree/03-01.html中。

5.4 修改节点的默认图标

实际上，每个树形节点都有icon和iconCls属性，它们负责指定节点的图标。

现在我们来修改树形中节点的图标。无论是通过new手工创建的节点，还是通过JSON读取到的节点，它们的设置方式都是一样的。

下面是使用icon的方法，如下面的代码所示。

```
var root = new Ext.tree.TreeNode({
    text: 'icon',
    icon: 'user_female.png'
});
```

下面是使用iconCls的方法，如下面的代码所示。

```
var node1 = new Ext.tree.TreeNode({
    text: 'iconCls',
    iconCls: 'icon-male'
});
```

使用iconCls，我们还需要在HTML中添加对应的CSS定义，如下面的代码所示。

```
.x-tree-node-leaf .icon-male {
    background-image: url(user_male.png)
}
```

这里有一个地方需要注意，iconCls对应的是icon-male，但写在CSS里要用层叠的写法定义.x-tree-node-leaf下的.icon-male，否则CSS会不起作用。

当然也可以同时使用iconCls和icon两种方法，如下面的代码所示。

```
var node2 = new Ext.tree.TreeNode({
        text: 'icon + iconCls',
        icon: 'user_female.png',
        iconCls: 'icon-male'
    });
```

两种方法都可以达到我们的目的（见图5-14）。不过，比较一下两种方法都使用的情况，到底哪种方法的优先级更高呢？

结果是icon获胜，这其实很好理解。iconCls只能定义背景图片，icon设置的是IMG的SRC部分，icon中设置的图片会把背景部分挡住。当然，实际使用时我们只会选择其中的一个。

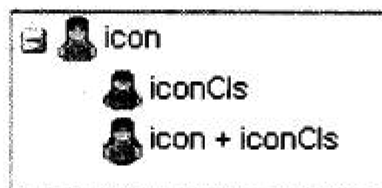


图5-14 自定义节点图标

示例在05.tree/04-01.html中。

5.5 从节点弹出对话框

实际上，我们从事件中获得的node只是一个对象，而不是HTML中的一个实际的DOM元素，所以不能直接用animEl:node来实现飞出效果。

EXT里的树节点都遵从MVC设计，所以要找到对应的DOM元素，应该使用节点的View部分。在TreeNode里，这部分叫做UI。树形节点的UI又包括好几部分，缩进用的空白、节点之间的连接线、节点展开和折叠的图标，以及显示的标题。

下面的代码可以让弹出的对话框看起来像是从标题上飞出来的，如图5-15所示。



图5-15 飞出来的对话框

```
tree.on("click", function(node){
    Ext.Msg.show({
        title: '提示',
        msg: "你单击了" + node,
        animEl: node.ui.textNode
    });
});
```

示例在05.tree/05-01.html中。

5.6 节点提示信息

节点提示信息的效果如图5-16所示。



图5-16 提示信息

当把鼠标停留在某个节点的上方时，便会出现提示信息。为了实现这种效果，要对提供的JSON数据做一些修改。在JSON中添加对应的节点提示内容，给每个节点数据都加上qtip:'xxx'参数，然后节点就可以显示提示信息了，如代码清单5-6所示。

代码清单5-6 提示信息

```
[
    {text:'01',qtip:'01',children:[
        {text:'01-01',qtip:'01-01',leaf:true},
        {text:'01-02',qtip:'01-02',children:[
            {text:'01-02-01',qtip:'01-02-01',leaf:true},
            {text:'01-02-02',qtip:'01-02-02',leaf:true}
        ]},
        {text:'01-03',qtip:'01-03',leaf:true}
    ]},
    {text:'02',qtip:'02',leaf:true}
]
```

不过，仅仅为树形的JSON添加这些参数还不能在页面上显示提示信息，需要先在JavaScript中对EXT的提示功能进行初始化。

```
// 开启提示功能
Ext.QuickTips.init();
```

上面这行代码必须在其他功能加载前完成，建议写在onReady函数的第一行。完整代码如下：

```
Ext.onReady(function(){

    // 开启提示功能
    Ext.QuickTips.init();
```

```
    var tree = new Ext.tree.TreePanel({
        el: 'tree',
        loader: new Ext.tree.TreeLoader({dataUrl: '06-01.txt'})
    });

    var root = new Ext.tree.AsyncTreeNode({
        id: '0',
        text: '我是根'
    });

    tree.setRootNode(root);
    tree.render();

    root.expand();

});
```

示例在05.tree/06-01.html中。

5.7 为节点设置超链接

虽然我们完全可以监听click事件，但是直接在节点树形中设置超链接的地址也是一个好主意，这是很多人都想实现的功能。

现在让我们给节点加上href属性，如下面的代码所示。

```
{text:'01-01',qtip:'01-01',leaf:true,href:'07-01.html'}
```

添加上面的代码后，点击01-01节点就会弹出03-10a.html页面。不过这种配置会修改当前页面，怎么才能打开另外一个窗口呢？大家可能会立即想到用target。不过很可惜，树形中的target参数名称另有他用，这里我们需要配置的参数名为hrefTarget，如下面的代码所示。

```
{text:'02',qtip:'02',leaf:true,href:'07-01.html',hrefTarget:'_blank'}
```

这样的href和hrefTarget组合确保我们可以在正确的地方打开正确的网页，所有参数都可以通过JSON获得。JSON数据如下所示：

```
[
    {text:'01',qtip:'01',children:[
        {text:'01-01',qtip:'01-01',leaf:true,href:'07-01.html'},
        {text:'01-02',qtip:'01-02',children:[
            {text:'01-02-01',qtip:'01-02-01',leaf:true,href:'07-01.html'},
            {text:'01-02-02',qtip:'01-02-02',leaf:true,href:'07-01.html'}
        ]},
        {text:'01-03',qtip:'01-03',leaf:true,href:'07-01.html'}
    ]},
    {text:'02',qtip:'02',leaf:true,href:'07-01.html',hrefTarget:'_blank'}
]
```

前端展示的代码只需在之前的例子基础上，将TreeLoader的dataUrl属性修改为07-01.txt即可。

示例在05.tree/07-01.html中。

5.8 直接修改树节点名称

从传统操作的角度来考虑，如果我们需要修改一个节点的名称，必须先选择这个节点，然后单击节点旁边的修改按钮，页面会跳转到一个修改页面。在这个修改页面中，我们修改对应的数据，然后再提交。在显示出一个操作成功的页面后，再次回到显示树的页面，这样才能看到修改后的结果。

其实我们只希望修改节点的标题，效果如图5-17所示。

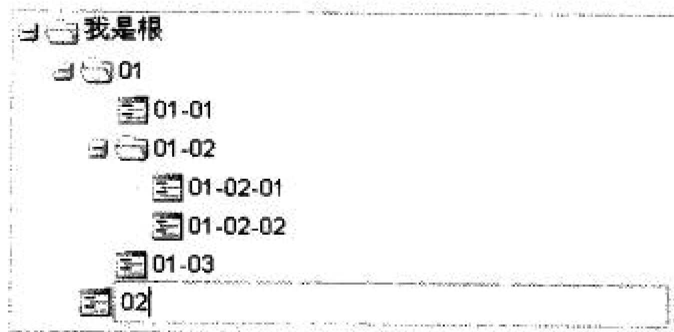


图5-17　编辑节点的文本内容

单击标题就可以修改，修改完了，按下Ctrl键就完成了整个修改过程。

代码非常简单，不需要大的修改，只需要为树形创建一个TreeEditor，再把TreePanel放进去即可，如下面的代码所示。

```
var treeEditor = new Ext.tree.TreeEditor(tree, {
    allowBlank    : false
});
```

接下来介绍一下上面示例中参数的作用。第一个参数tree就是刚才提到的TreePanel。allowBlank属于数据校验部分，这个属性和前面提到的TextField等控件的allowBlank属性的作用一样，都是指输入值不允许为空。至于为什么不能为空，我们可以想象一下，一个没有名字的节点会是什么样子。

这就是我们需要的所有东西，有了它们就可以自由修改树形节点的标题了。完整的代码如下所示：

```
var tree = new Ext.tree.TreePanel({
    el: 'tree',
    loader: new Ext.tree.TreeLoader({dataUrl: '06-01.txt'})
});

var treeEditor = new Ext.tree.TreeEditor(tree, {
    allowBlank: false
});

var root = new Ext.tree.AsyncTreeNode({
    id: '0',
    text: '我是根'
});

tree.setRootNode(root);
```

```
tree.render();

root.expand();
```

示例在05.tree/08-01.html中。

我们通过绑定事件还可以对TreeEditor实现更多控制，相关的事件列举如下。

- on("beforestartedit")：这个事件让我们控制是否允许编辑当前节点。我们用自己的方式判断node的属性，如果满足某个条件，比如leaf:false，就不允许编辑。操作很简单，返回false即可，如下面的代码所示。

  ```
  treeEditor.on("beforestartedit", function(treeEditor){
      return treeEditor.editNode.isLeaf();
  });
  ```

  在枝干节点和叶子节点上单击就可以看到不同的效果。

- on("complete")：完成修改后，按下Enter键时就会产生这个事件。我们可以在监听函数中得到修改后的数据，如下面的代码所示。

  ```
  treeEditor.on("complete", function(treeEditor){
      alert(treeEditor.editNode.text);
  });
  ```

这里的editNode.text是我们修改之前的结果。

示例在05.tree/08-02.html中。

5.9 树形的拖放

如果想让树的节点可以自由拖动（见图5-18），创建TreePanel时设置enableDD:true即可。不过，直接设置enableDD:true属性只能实现叶子与枝干和根之间的拖放，叶子不能拖放到叶子下。完整的代码如下：

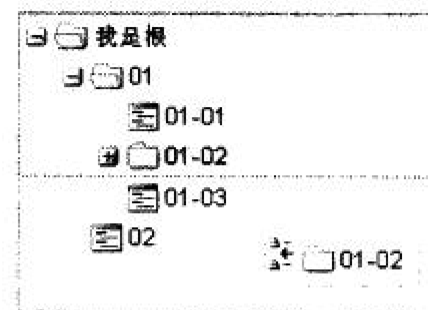


图5-18 拖放树形节点

```
var tree = new Ext.tree.TreePanel({
    el: 'tree',
    enableDD: true,
    loader: new Ext.tree.TreeLoader({dataUrl: '06-01.txt'})
});

var root = new Ext.tree.AsyncTreeNode({
    id: '0',
    text: '我是根'
});

tree.setRootNode(root);
tree.render();

root.expand();
```

之后的小节里的拖放都是在上面的代码中添加相应的拖放事件和参数配置。

示例在05.tree/09-00.html中。

5.9.1　节点拖放的3种形式

当然，拖（drag）就是把一个节点拖起来，拖哪个节点都是一样的。不过放（drop）就复杂些，所以说EXT为拖放提供3种形式还是有道理的。

- append。放下去的节点会变成被砸中的节点的子节点，最后形成的是父子关系。append的标记是一个绿色的加号图标，如图5-19所示。
- above。放下去的节点与目标节点是兄弟关系，放下去的节点排行在前。above的标记是一个箭头和两个短横线，如图5-20所示。
- below。放下去的节点与目标节点还是兄弟关系，不过这次放下去的节点排行在后。below的标记图标和above的标记图标恰好相反，如图5-21所示。

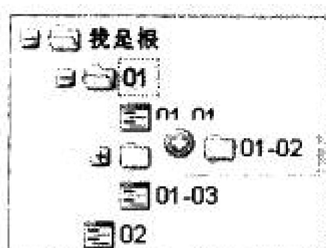


图5-19　append拖放



图5-20　above拖放



图5-21　below拖放

TreePanel中有很多事件与拖放有关，通过它们，我们可以把节点从树的顶端拖到底端。上面我们只介绍了几个最常用的。

5.9.2　叶子不能 append

在拖放时有没有发现叶子节点不能append？这是因为EXT内部有一些限制，如果节点包含leaf:true参数，就不能用拖放的方式添加子节点。这个限制明显不合理，但为了保持版本稳定，最好不要修改源代码，因此我们还是使用事件解决这个问题。

我们需要的事件是nodedragover，当你拖动个某个节点经过另外一个节点的顶部时就会触发这个事件。当这个节点是叶子时，我们就可以进行下面的操作，从而突破EXT不允许在叶子节点上添加子节点的限制，如下面的代码所示。

```
// 拖放判断
tree.on("nodedragover", function(e){
    var n = e.target;
    if (n.leaf) {
        n.leaf = false;
    }
    return true;
});
```

事件发生以后，会向绑定的函数发送一个参数e，e.target是当前鼠标指针经过的节点。我们先判断n.leaf是否为true，如果为true，就说明这个节点是叶子节点，对它的操作就是让leaf属性变成false，然后就可以为这个节点添加子节点了。

示例在05.tree/09-02-01.html中。

5.9.3 判断拖放的目标

nodedrop事件是拖放的节点放下去时触发的，如下面的代码所示。

```
tree.on("nodedrop", function(e){
    Ext.Msg.alert('提示', '我们的节点' + e.dropNode + '掉到了' + e.target + '上,
      掉落方式是' + e.point);
});
```

这里充分利用了事件发生时传递过来的参数：e.dropNode是正在拖放的节点，e.target是放下去碰到的节点，e.point是放下去的方式（append、above、below）。

通过这些数据，我们就可以知道当前节点的位置和状态，从而计算出数据并通过Ajax发送给后台，让后台对节点的数据进行更新。下面我们来实践一下，再复习一下EXT里如何使用Ajax传输数据。在原来拖放代码的基础上添加Ajax部分的代码，如代码清单5-7所示。

代码清单5-7 拖放事件

```
tree.on("nodedrop", function(e){
    Ext.Msg.alert('提示', '我们的节点' + e.dropNode + '掉到了' + e.target + '上,
                   掉落方式是' + e.point);
    var item = {
        dropNode: e.dropNode.id,
        target: e.target.id,
        point: e.point
    };
    Ext.lib.Ajax.request(
        'POST',
        '09_03_01.jsp',
        {success: function(response){
            Ext.Msg.alert('信息', response.responseText);
        },failure: function(){
            Ext.Msg.alert("错误", "与后台联系时出现了问题");
        }},
        'data=' + encodeURIComponent(Ext.encode(item))
    );
});
```

实际上，这里的Ext.lib.Ajax并非EXT的一部分。为什么这样说呢？之前我们也提到过，EXT是基于其他JavaScript基础工具库建立起来的组件库，很多底层功能都会利用这些JavaScript基础工具库的功能，Ajax就是其中之一。不信你可以去找找，Ext.lib.Ajax的定义写在ext-base.js里，而不是ext-all.js里。如果你使用了其他的适配器，Ext.lib.Ajax就定义在其他的适配器中。只要这个Ext.lib.Ajax的名字不变，具体的内部实现都是由你选择的适配器决定的。

如果你对底层没有太大兴趣，只要会用这个Ext.lib.Ajax就可以了，在此我们只介绍它的用法。

我们调用了request()函数，它包含4个参数。请注意，包含success和failure的一大串JSON对象其实只是一个参数。

第一个参数：'POST'代表HTTP协议的方法。如果你对HTTP协议并不了解，请选择'POST'，这样可以避免中文参数乱码和缓存方面的问题。

第二个参数：'09_03_01.jsp'是请求的URL。

第三个参数被称为回调函数。之所以叫回调函数，是因为这里定义的函数会在Ajax执行成功或失败后被调用。{success:fn,failure:fn}里包含两个值，success对应的函数会在请求成功后调用，而failure对应的函数会在请求失败后调用。

这里的成功和失败与数据库访问的成功或失败完全不一样，不要用业务标准来判断操作是否成功。Ajax的成功或失败在于HTTP响应的状态码是否等于200。

或许有人还不了解HTTP，在此简单介绍一下。当使用浏览器访问页面时，如果没有返回“404找不到页面”或“500服务器内部错误”等信息，就表明请求成功。Ajax只能分辨出这种响应错误，至于后台的业务操作是否成功，就需要自己编写代码根据后台返回的具体信息进行判断了。

示例里的failure很简单，只弹出错误提示。success则稍微复杂一些，因为它有一个response参数。这个response参数表示响应对象，其中包含响应状态和响应内容等信息，EXT帮我们把这些都封装好了。例如，我们这里通过response.responseText获得的就是文本形式的返回信息。如果需要XML格式的返回信息，可以使用response.responseXML，大家可以自己试一下。

第四个参数是发送给后台的参数，格式为name1=value1&name2=value2。至于为什么用encodeURIComponent(Ext.encode(item))，一方面是因为Ext.encode()会把JSON转换成字符串，另外一方面是为了对参数进行编码，避免出现非法字符，这是通用方式。

09_03_01.jsp的后台脚本（见代码清单5-8）非常简单，只是获得了参数，然后直接返回给前台。实际环境中，需要先对字符串进行解析，获得自己需要的数据并进行处理。这里只是演示，就不详细讲解了。

代码清单5-8　处理事件的后台JSP代码

```
<%@ page contentType="text/html;charset=utf-8"%>
<%
    request.setCharacterEncoding("UTF-8");
    response.setCharacterEncoding("UTF-8");

    String data = request.getParameter("data");
    System.out.println(data);

    response.getWriter().print(data);
%>
```

最后的结果就是这样，每次拖放都会向后台发送数据，然后前台显示返回的响应结果（见图5-22）。

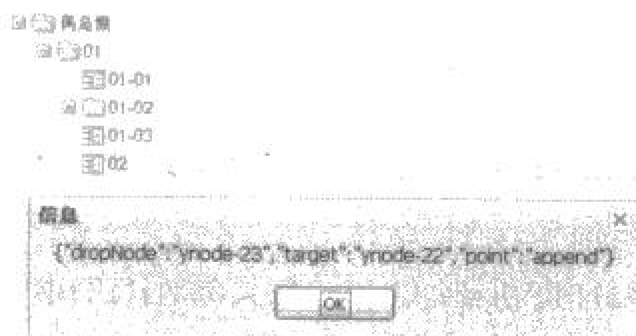


图5-22　拖放事件

示例就在05.tree/09-03-01.html中。

5.9.4 树之间的拖放

我们可以在一棵树内部进行拖放，也可以在两棵树之间实现拖放。但drag和drop并不一定同时存在，既可以使用enableDD:true，也可以单独使用enableDrag或enableDrop分别指定一棵树支持drag或是drop。如此一来，可以限制用户只能从一棵树上拖到另一棵树上，如代码清单5-9所示。

代码清单5-9 树之间的拖放

```
var tree1 = new Ext.tree.TreePanel({
    el: 'tree1',
    enableDrag: true,
    loader: new Ext.tree.TreeLoader({dataUrl: '06-01.txt'})
});

var tree2 = new Ext.tree.TreePanel({
    el: 'tree2',
    enableDrop: true,
    loader: new Ext.tree.TreeLoader({dataUrl: '06-01.txt'})
});
```

参考上面的设置，在tree1里使用enableDrag，tree2里使用enableDrop，用户只能将tree1里的节点拖到tree2上。我们禁用了树本身的拖放，也不允许将tree2中的节点拖放到tree1里。

示例在05.tree/09-04-01.html中。

注意 树之间的拖放最好只是将tree1的枝和叶拖放到tree2上去，如果将根也拖过去的话会出错。虽然把根拖2次也能拖过去，但是不建议这样做。

5.10 树形过滤器 TreeFilter

如果树的节点过多，用户可能需要执行搜索操作，从而过滤掉不需要的节点。这时需要用到Ext.tree.TreeFilter，可以使用它提供的函数判断哪些节点需要显示，哪些节点不应该显示。

刚打开页面时，我们会看到树中显示了所有的节点（见图5-23），按下“只显示‘02’”按钮就会看到图5-24所示的结果。



图5-23 过滤之前



图5-24 过滤之后

10-01.html中的JavaScript代码如代码清单5-10所示。

代码清单5-10　树形过滤器

```
var tree = new Ext.tree.TreePanel({
    loader: new Ext.tree.TreeLoader({
        dataUrl: '10-01.txt'
    }),
    root: new Ext.tree.AsyncTreeNode({
        id: '0',
        text: '根'
    }),
    autoHeight: true,
    renderTo: 'tree'
});

tree.expandAll();

var treeFilter = new Ext.tree.TreeFilter(tree, {
    clearBlank: true,
    autoClear: true
});

Ext.get('clearAll').on('click', function() {
    treeFilter.clear();
});
Ext.get('select').on('click', function() {
    treeFilter.filter('02');
});
```

同时在html页面中创建两个按钮，代码如下所示：

```
<button id="clearAll">全部显示</button>
<button id="select">只显示"02"</button>
```

JavaScript代码中的new Ext.tree.TreeFilter(tree,{})将TreePanel当作参数传入TreeFilter。执行这样的绑定之后，通过TreeFilter执行的任何操作都可以直接反应到TreePanel上。第二个参数是TreeFilter所需的附加参数，这个参数会在下面讨论。

下面，我们要监听两个按钮的单击事件。单击第一个按钮时，执行treeFilter.clear()清空过滤条件，这就会让树形显示出所有节点；单击第二个按钮会过滤所有名称中包含'02'内容的节点。注意，它会先过滤上层节点，如果上层节点不符合条件，那么它不会去过滤该层节点的子节点。

现在我们来看一下TreeFilter中使用的参数。

- clearBlank：如果查询的字符串是空字符串，就执行clear()。
- reverse：这个参数用于反向操作过滤条件，就是在filter('data')时显示所有不符合过滤条件的节点。但是官方代码中存在一个问题，为了让它可以正常运行，需要把TreeFilter.js的第77行中的“if(!m || rv){”改成“if(!(m^rv)){”。
- autoClear：每次过滤之前先执行clear()，否则会在上次过滤结果的基础上进行查询。

- remove：会删除不符合过滤条件的节点，这样就不能使用clear()恢复为过滤之前的状态了。

最后来看一段有用的代码，它来自docs/resources/docs.js（运行结果如图5-25所示），可以根据用户输入的条件对树进行过滤。

如果我们查询“-02”，就会显示包含“-02”的节点，如图5-26所示。

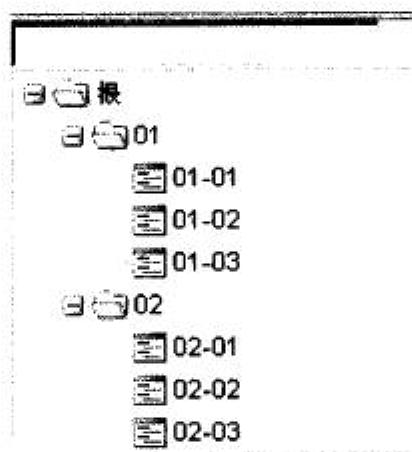


图5-25 模糊过滤之前

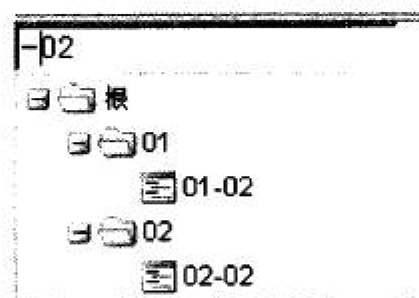


图5-26 模糊过滤之后

这个示例有如下3大特点。

- 监听输入框的keyup事件，每次按键之后都会重新过滤树形中的节点，如下面的代码所示。

```
// 按键后触发事件
field.on('keyup', function(e) {
    var text = field.dom.value;
}
```

- 进行过滤时会自动避开非叶子节点，这样才可以得到所有匹配的叶子节点，如下面的代码所示。

```
// 根据输入编写一个正则表达式，'i'代表不区分大小写
var re = new RegExp(Ext.escapeRe(text), 'i');
filter.filterBy(function(n){
    // 只过滤叶子节点，避免枝干被过滤时，底下的叶子都无法显示
    return !n.isLeaf() || re.test(n.text);
});
```

- 在把不匹配的叶子节点隐藏起来后，还要把空的非叶子节点隐藏起来，如下面的代码所示。

```
// 如果这个节点不是叶子，而且下面没有子节点，就应该隐藏起来
hiddenPkgs = [];
tree.root.cascade(function(n) {
    if(!n.isLeaf() && n.ui.ctNode.offsetHeight < 3){
        n.ui.hide();
        hiddenPkgs.push(n);
    }
});
```

完整示例在05.tree/10-02.html中，也可以直接查看docs/resourcs/docs.js源文件。

5.11 利用 TreeSorter 对树进行排序

TreeSorter是一个专门用来对树节点进行排序的工具。例如，你获得了一堆散乱的节点信息，就可以用TreeSorter把它们排列整齐。

现在有这样一棵散乱的树，如图5-27所示。

通过TreeSorter处理之后，这些节点就会变得很整齐了（见图5-28）。

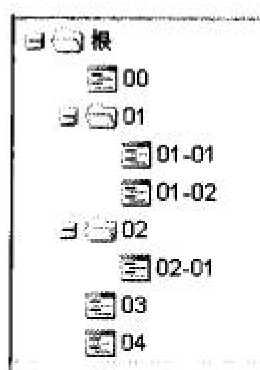


图5-27 排序之前

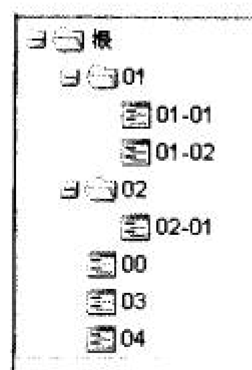


图5-28 排序之后

实现的代码很简单，仅使用了一个folderSort参数，如下面的代码所示。

```
var sorter = new Ext.tree.TreeSorter(tree, {
    folderSort: true,
});
```

folderSort参数可以让所有叶子节点排在非叶子节点后面。下面介绍在TreeSorter中使用的其他几个参数。

- caseSensitive：是否区分大小写，默认为false，不区分大小写。
- dir：排序方式（asc/desc），默认为asc，升序排列。
- folderSort：将叶子节点排到非叶子节点后面，默认为false。
- leafAttr：判断叶子节点的标志，默认为leaf。如果node中存在leaf:true参数，就认为这个节点是叶子。
- property：根据节点的属性进行排序，默认为text。
- sortType：这是在排序前对数据进行预处理的函数，默认情况下是直接从node中取得text属性n.attributes('text')。

示例在05.tree/11-01.html中。

5.12 树形节点视图——Ext.tree.TreeNodeUI

Ext.tree.TreeNodeUI是生成Ext.tree.TreeNode实例时默认使用的视图组件。在生成每个Ext.tree.TreeNode实例时，它会先查找this.attributes.uiProvider和this.defaultUI。如果有任何一个属性存在，它就会使用这个属性创建自己的视图。如果这两个属性都不存在，就会使用Ext.tree.TreeNodeUI创建视图。因此，在树形结构中通常都是使用Ext.tree.TreeNodeUI作为树形节点的视图。

可以通过node.ui的方式获得某个Ext.tree.TreeNode实例对应的Ext.tree.TreeNodeUI，如下面的代码所示。

```
tree.on('click', function(node) {
    var ui = node.ui;
    ui.addClass("big");
    (function() {
        ui.removeClass("big");
    }).defer(1000);
});
```

上例中监听Ext.tree.TreePanel的click事件。当用户单击一个节点时，会获得节点对应的Ext.tree.TreeNodeUI实例，之后还会调用实例的addClass()函数将节点的文字转换为粗体显示，并在1秒之后调用removeClass()将刚刚设置的CSS样式清除掉。单击树形后的效果如图5-29所示。完整代码如下所示：

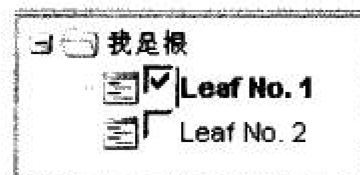


图5-29 修改Ext.tree.TreeNodeUI的文字样式

```
var tree = new Ext.tree.TreePanel({
        el: 'tree',
        loader: new Ext.tree.TreeLoader()
    });

    var root = new Ext.tree.AsyncTreeNode({
        text:'我是根',
        children: [
            {text: 'Leaf No. 1',leaf: true, checked: true},
            {text: 'Leaf No. 2',leaf: true, checked: false}
        ]
    });

    tree.setRootNode(root);
    tree.render();

    root.expand();

    tree.on('click', function(node) {
        var ui = node.ui;
        ui.addClass("big");
        (function() {
            ui.removeClass("big");
        }).defer(1000);
    });
```

除了addClass()和removeClass()函数之外，Ext.tree.TreeNodeUI还提供了几类功能函数，如下所示。

- getAnchor()、getIconEl()、getTextEl()这3个函数可以分别获得页面上与树形对应的<a>标签，包含图标的<img>标签和包含文字的<span>标签部分。
- hide()和show()函数可以控制树形节点是否隐藏。
- isChecked()和toggleCheck()函数可以判断和修改树形节点中Checkbox的状态。不过

这两个属性也只有在属性节点中包含Checkbox时才可以使用，否则isChecked()会一直返回false。

Ext.tree.TreeNodeUI中的功能函数的应用如下面的代码所示。

```
tree.on('click', function(node) {
    var ui = node.ui;
    alert(ui.getAnchor() + "," + ui.getIconEl() + "," + ui.getTextEl());
    alert(ui.isChecked());
    ui.toggleCheck();
});
```

5.13 表格与树形的结合——Ext.ux.tree.ColumnTree

对于某些场景，我们需要在表格中实现分级显示的功能，它与分组表格Ext.grid.GroupingGrid的风格有些类似，两者都支持对不同分类数据执行展开和折叠的操作。在EXT中可以通过扩展Ext.tree.TreePanel和Ext.tree.TreeNodeUI的方式实现这种表格与树形结合的效果，如图5-30所示。

示例

任务	持续时间	负责人
EXT程序开发	12个月	
第一章介绍EXT	1个月	Lingo
第二章GRID	2个月	ForgetDavid
第十一章常用工具	9个二月	kayzhan
协同办公OA	3个月	
平台搭建	1个月	kayzhan
系统设计	1个月	kayzhan
编码实现	1个月	kayzhan

图5-30 表格和树形的结合

Ext.ux.tree.ColumnTree其实属于EXT的扩展件。为了在应用中使用Ext.ux.tree.ColumnTree，我们首先需要引用EXT发布包中的column-tree.css、ColumnNodeUI.css以及ColumnNodeUI.js三个文件。因为Ext.tree.ColumnTree目前只是以扩展的形式放在examples下的tree目录下，它还没有被包含到EXT核心库中，所以我们需要找到这两个文件，以手工的方式导入到我们自己的应用中，如下面的代码所示。

```
<script type="text/javascript" src="../ux/ColumnNodeUI.js"></script>
<link rel="stylesheet" type="text/css" href="../ux/css/ColumnNodeUI.css" />
<link rel="stylesheet" type="text/css" href="../tree/column-tree.css" />
```

在引用了ColumnNodeUI.js之后，我们就可以使用Ext.ux.tree.ColumnTree创建表格形式的树形了，如下面的代码所示。

```
var tree = new Ext.ux.tree.ColumnTree({
    width: 550,
```

```
    height: 300,
    rootVisible:false,
    autoScroll:true,
    title: '示例',
    renderTo: 'tree',

    columns:[{
        header:'任务',
        width:330,
        dataIndex:'task'
    },{
        header:'持续时间',
        width:100,
        dataIndex:'duration'
    },{
        header:'负责人',
        width:100,
        dataIndex:'user'
    }],

    loader: new Ext.tree.TreeLoader({
        dataUrl:'_01_04-12.txt',
        uiProviders:{
            'col': Ext.ux.tree.ColumnNodeUI
        }
    }),

    root: new Ext.tree.AsyncTreeNode({
        text:'Tasks'
    })
});
```

上述代码中，有以下两点与一般情况下创建树形时使用的参数不一致。

- columns参数的值与我们使用Ext.grid.GridPanel时完全相同，这里使用columns参数来指定每一行应该分为几列进行显示。我们这里定义了3列，分别是“任务”、“持续时间”和“负责人”，这3列分别对应着后台数据中的“task”、“duration”和“user”。
- 创建Ext.tree.TreeLoader时，在uiProviders参数中配置了ui:Ext.ux.tree.Column-NodeUI，这样我们就可以在生成树形节点时选择使用Ext.ux.tree.ColumnNodeUI生成节点视图。

前台脚本已经准备完毕，我们还需要为前台供应匹配的后台数据，如下面的代码所示。

```
[{
    task:'深入浅出ExtJs',
    duration:'12个月',
    user:'',
    uiProvider:'col',
    cls:'master-task',
```

```
        iconCls:'task-folder',
        children:[{
            task:'第1章  EXT概述',
            duration:'1个月',
            user:'Lingo',
            uiProvider:'col',
            leaf:true,
            iconCls:'task'
        },{
            task:'第2章  EXT框架基础',
            duration:'2个月',
            user:'ForgetDavid',
            uiProvider:'col',
            leaf:true,
            iconCls:'task'
        },{
            task:'第11章  实用工具',
            duration:'9个月',
            user:'kayzhan',
            uiProvider:'col',
            leaf:true,
            iconCls:'task'
        }]
    },{
```

我们从后台提供的数据中摘录一部分进行讨论。与之前用到的树形数据相同，后台提供的是一个JSON 数组，我们为数组中的每个对象都指定了task、duration和user这3个属性。这3个属性会根据前台脚本中设置的columns参数对应显示在“任务”、“持续时间”、“负责人”这3列的下面。之前讨论过的cls、iconCls、leaf、children等属性的作用没有变，我们在这里依旧使用cls和iconCls来定义树形节点的样式和图标，使用children为枝干节点设置子节点，使用leaf定义叶子节点。

需要注意的是，我们需要为每个节点对象设置uiProvider:'col'，这样在生成树形时Tree-Loader会从预先定义的uiProviders参数中取得'col'对应的Ext.ux.tree. ColumnNodeUI，用它来作为显示树形节点的视图，这样我们才能得到期望中的表格属性。

在新发布的EXT 3.1中添加了另一个树形和表格相结合的扩展组件TreeGrid，详细介绍可以参考14.5.3节。

5.14 小结

本章讨论了EXT中树形的原理和使用方法，包括创建树形、使用本地数据和远程JSON为树形添加节点，以及TreeLoader的部分配置方式。

同时讲解了树形事件，包括如何在树形中使用右键菜单、修改节点图标、为节点设置提示信息、为节点设置超链接、直接修改树形节点名称等内容。

此外，集中讨论了树形拖放，介绍了树形多种拖放方式，以及树形过滤器和使用TreeSorter对树形中的节点进行排序，并在最后一节中讲述了表格与树形结合的扩展件。

第6章 拖放

本章内容

- 拖放简介
- 拖放的简单应用
- 拖放组件体系
- 拖放的事件
- 高级拖放

6

6.1 拖放简介

拖放在EXT的组件体系中占有很重要的地位，很多组件内部都封装了对拖放功能的实现。但是，EXT官方却几乎没有提供单独使用拖放功能的示例。本章将主要讨论拖放的组件结构和范例，重点是单独使用Ext.dd包实现拖放功能，而不是与其他组件相结合。

EXT中的拖放有如下功能。

- 可以拖放表格里的行，让它们按指定的方式排列。
- 可以拖放树里的节点，把节点从一个枝干拖向另一个枝干。
- 在表格和树之间也可以进行拖放。
- 布局的split也是一种拖放，可以改变布局的大小。
- resize也算是拖放，改变大小。

6.2 拖放的简单应用

对于B/S系统而言，拖放一直是一项很少用到的功能。我们一直认为实现拖放功能很复杂，但是在EXT中只需要一行代码就可以实现最基本的拖放功能，如下面的代码所示。

```
new Ext.dd.DDProxy('block');
```

对应的HTML部分的代码如下所示。

```
<div id="block" style="background: red;"> </div>
```

如果不进行任何修饰，根本无法看到拖放的效果，因此我们为block加上了红色的背景色（见图6-1）。现在我们可以任意拖动这条红色的div，体验EXT为我们提供的拖放功能了。

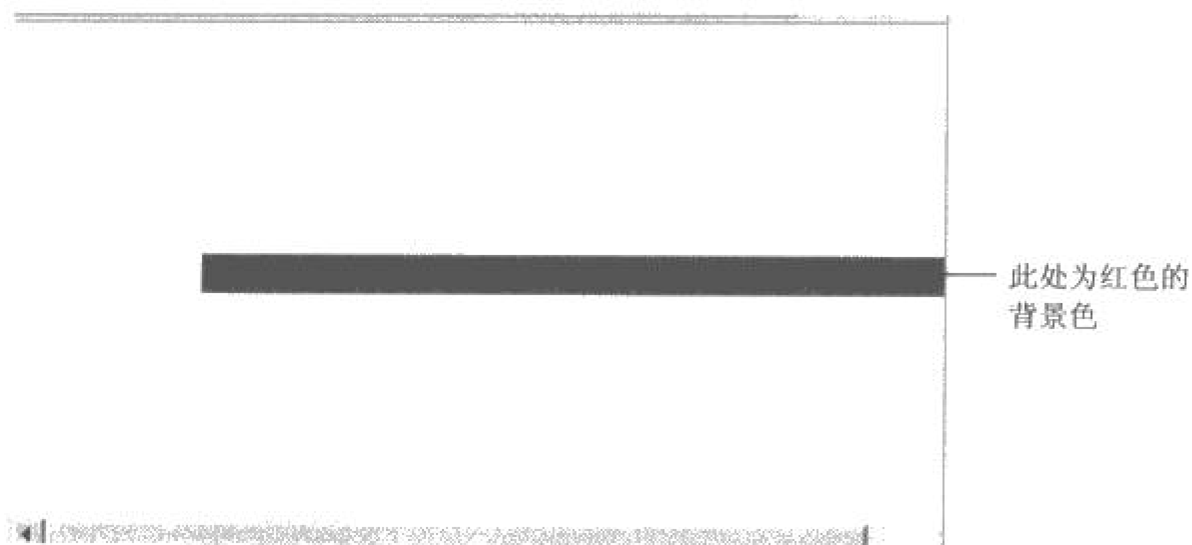


图6-1　最简单的拖放

该示例在06.dnd/01.html中。

6.3　拖放组件体系

我们先看一下拖放的继承关系图（见图6-2），这有助于我们的理解。

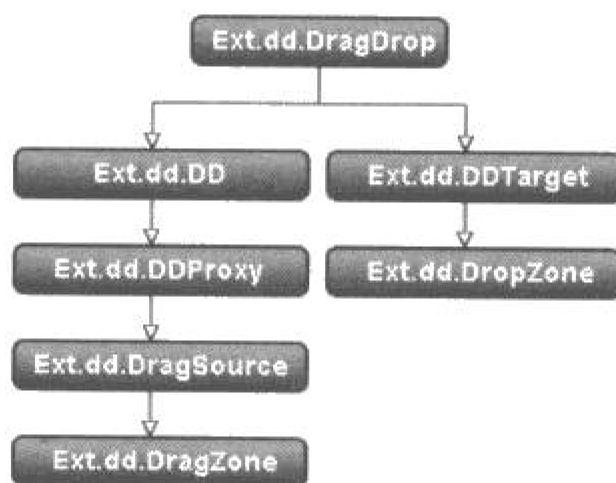


图6-2　拖放的继承关系图

简单来说，图6-2左边的4个组件都是可以随意拖动的。拖动起来以后，直接把它们放到右边那些定义好的区域中。proxy表示可拖动对象，target表示拖动的目的地。

我们看了上面的继承图（图6-2），并对它有了简单的了解。接下来看看下面的示例，其中的proxy是可以任意拖放的，如下面的代码所示。

```
var proxy = new Ext.dd.DragSource('proxy', {group:'dd'});
```

target告诉我们可以把上面的proxy放到哪些地方，如下面的代码所示。

```
var target = new Ext.dd.DDTarget('target', 'dd');
```

注意到两者之间相同的dd了吗？这是分组标志，用来限制拖放的目的地。只有相同组名的目的地才能接收它，好比超市中货架上的商品都是放在指定区域一样。

不过，这也只是拖放而已，没有任何其他效果。下面我们加入一些小技巧，可以让拖放显得更神奇一些，如下面的代码所示。

```
proxy.afterDragDrop = function(target, e, id) {
    var destEl = Ext.get(id);
    var srcEl = Ext.get(proxy.getEl());
    srcEl.insertBefore(destEl);
};
```

从上面的代码中可以看出，拖放后会执行上面代码中的对应函数，并通过id获得target，然后根据proxy.getEl()获得proxy，最后把proxy添加到target中。上述代码运行后的页面效果如图6-3所示。

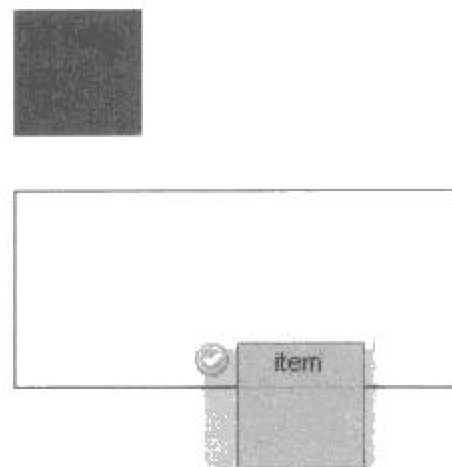


图6-3 拖放效果

当然，为了让拖放效果更清晰明了，我们加入了很多CSS样式，如代码清单6-1所示。

代码清单6-1 实现更清晰拖放效果的CSS代码


```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=gbk">
        <title>dd</title>
        <link rel="stylesheet" type="text/css" href="../../resources/css/ext-all.css" />
        <script type="text/javascript" src="../../adapter/ext/ext-base.js"></script>
        <script type="text/javascript" src="../../ext-all.js"></script>
        <style type="text/css">
HR {
    clear: both;
    visibility: hidden
}
.block {
    border: 1px red solid;
    margin: 10px;
    min-height: 80px;
}
.item {
    border: 1px green solid;
    background: green;
    float: left;
    margin:10px;
    width: 50px;
    min-height: 50px;
    text-align: center;
}
        </style>
        <script type="text/javascript">
Ext.onReady(function(){

    var proxy = new Ext.dd.DragSource('proxy', {group:'dd'});

    proxy.afterDragDrop = function(target, e, id) {
```

```
            var destEl = Ext.get(id);
            var srcEl = Ext.get(proxy.getEl());
            srcEl.insertBefore(destEl);
        };

        var target = new Ext.dd.DDTarget('target', 'dd');

});
            </script>
        </head>
        <body>
            <script type="text/javascript" src="../examples.js"></script>
            <div id="proxy" class="item">item</div>
            <hr />
            <div id="target" class="block">
                <hr />
            </div>
        </body>
    </html>
```

该示例在06.dnd/02.html中。

6.4　拖放的事件

拖放相关的类都继承自Ext.dd.DragDrop，在DragDrop中定义了一系列拖放操作过程中需要使用的事件函数，我们可以通过这些事件函数对整个拖放过程进行控制。

此处提到的事件与EXT事件模型并没有任何关系，它们是专门用于处理拖放的函数。如startDrag、onDrag、onDragDrop、endDrag、onDragEnter、onDragOut、onDragOver、onInvalidDrop、onMouseDown和onMouseUp，这些函数分别代表了不同阶段的拖放过程。在实际使用中，我们需要重写对应的事件函数，从而监听和处理拖放功能。

上述的事件函数可以分为以下3大类。

- startDrag、onDrag、onDragDrop、endDrag是第一类拖放事件函数，它们构成了拖放的主要过程（见图6-4）。
  - startDrag(int x,int y)：开始拖放事件，参数是包含x、y的坐标值。
  - onDrag(Event e)：正在拖放事件，在拖放一个对象时触发，参数是mousemove鼠标移动事件。
  - onDragDrop(Event e, String|DragDrop[] id)：正在放下事件，在一个对象放到另一个DragDrop对象上时触发，第一个参数是mouseup鼠标放开事件，第二个参数表示drop的目标位置。如果是在POINT模式下，就会传入目标元素的id；如果是在INTERSECT模式下，则会传入放下目标的拖放对象数组。

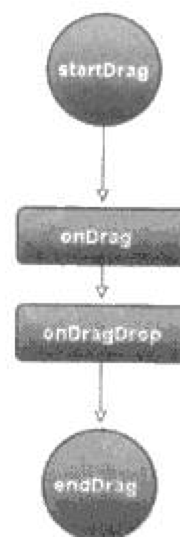


图6-4　第一类拖放事件函数

  - endDrag(Event e)：拖放结束事件，在拖放操作结束之后触发，参数是mouseup鼠标放开事件。

- onDragEnter、onDragOut、onDragOver、onInvalidDrag是第二类事件（见图6-5），它们细化了用户拖动对象的过程。
  - onDragEnter(Event e, String|DragDrop[] id)：进入drop区域事件，拖放过程中首次触碰到drop区域时触发。第一个参数是mousemove鼠标移动事件，第二个参数表示drop的目标位置。如果是在POINT模式下，就会传入目标元素的id；如果是在INTERSECT模式下，则会传入放下目标的拖放对象数组。
  - onDragOut(Event e, String|DragDrop[] id)：离开drop区域事件，拖放过程中从drop区域离开时触发。第一个参数是mousemove鼠标移动事件，第二个参数表示drop的目标位置。如果是在POINT模式下，就会传入目标元素的id；如果是在INTERSECT模式下，则会传入放下目标的拖放对象数组。

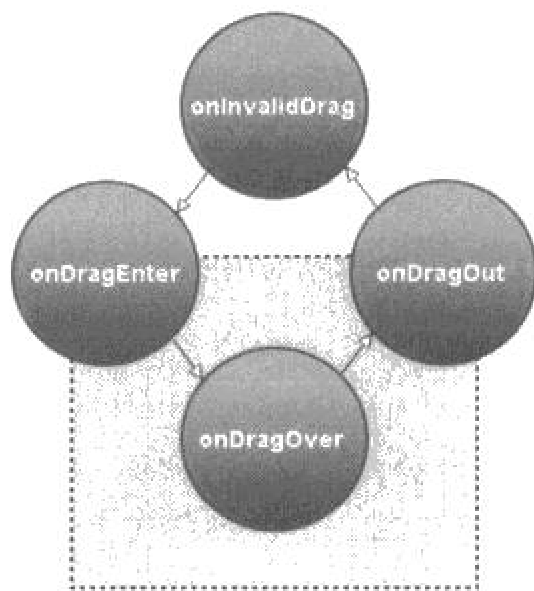


图6-5 第二类拖放事件函数

  - onDragOver(Event e, String|DragDrop[] id)：在drop区域中拖放移动事件，当在drop区域拖放移动时触发。第一个参数是mousemove鼠标移动事件，第二个参数表示drop的目标位置。如果是在POINT模式下，就会传入目标元素的id；如果是在INTERSECT模式下，则会传入放下目标的拖放对象数组。
  - onInvalidDrop(Event e)：不能drop事件，不在drop区域移动时触发，参数是mousemove鼠标移动事件。

- onMouseDown和onMouseUp是第三类拖放事件函数，它们用于对原始鼠标事件进行提示，而且已经融合在前两类拖放事件函数中了，它们的关系如图6-6所示。

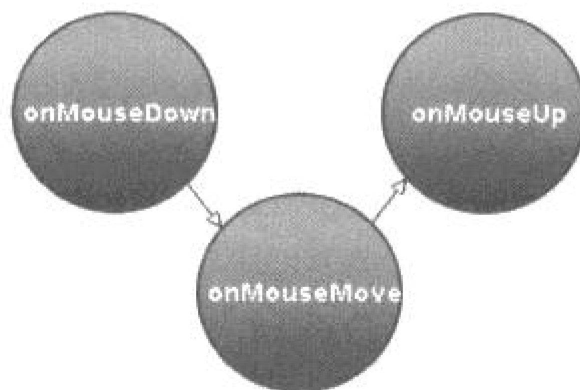


图6-6 第三类拖放事件函数

  - onMouseDown(Event e)：鼠标按下事件，参数是mousedown鼠标按下事件。
  - onMouseUp(Event e)：鼠标放开事件，参数是mouseup鼠标放开事件。

6.5 高级拖放

在EXT中，虽然拖放被单独放在一个命名空间中，但是examples目录下却没有关于拖放的示例。这让我们在享受表格和树的内置拖放功能的同时，又觉得很无奈。如何单独使用Ext.dd呢？幸运的是，YUI[①]为我们提供了大量拖放的示例，我们下面将逐一对这些示例进行讲解。

6.5.1 基础

我们先看看YUI中包含的基础（Basic）示例，这个示例是最基本的，代码如下所示。

```
dd1 = new Ext.dd.DD("dd-demo-1");
dd2 = new Ext.dd.DD("dd-demo-2");
dd3 = new Ext.dd.DD("dd-demo-3");
```

这部分是JavaScript代码，创建3个拖动对象，参数对应HTML中的3个id，让这3个对象变成可拖动的元素，如下面的代码所示。

```
<div id="dd-demo-1" class="dd-demo"></div>
<div id="dd-demo-2" class="dd-demo"></div>
<div id="dd-demo-3" class="dd-demo"></div>
```

class部分不用细讲，这些CSS样式只是为了让外观更漂亮，唯一需要注意的是div中的id。这3个div现在已经被EXT改造成可拖放的对象了，可以任意地拖放。功能很简单，它的效果如图6-7所示。

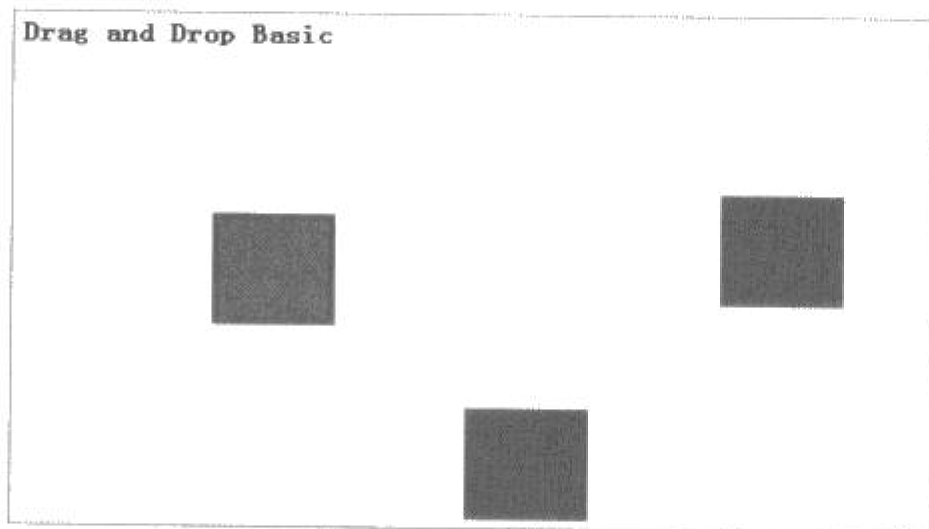


图6-7 基础（Basic）示例效果图

示例在06.dnd/04-01.html中。

6.5.2 控制柄

控制柄（Handle）就好比菜刀的刀柄，握着刀柄我们才能方便地使用菜刀。这里演示的是为拖放对象指定一个区域，用户只有抓住这个区域才能拖动整个对象（见图6-8）。

① EXT从YUI发展而来。

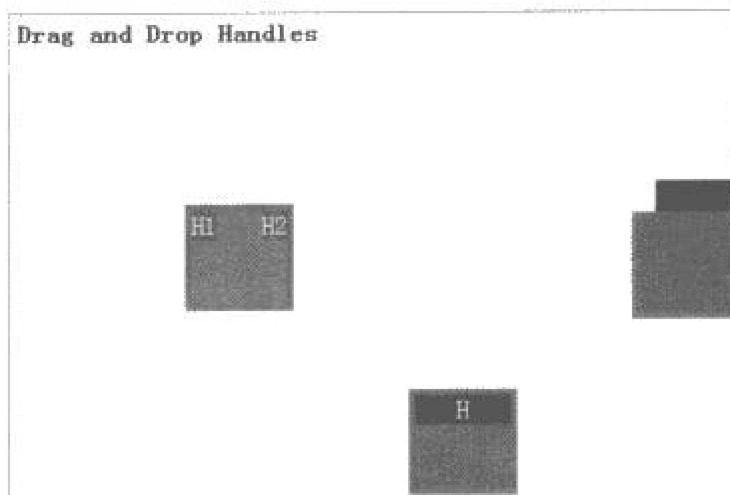


图6-8 控制柄（Handle）示例效果图

还是刚才的3个div，第一个div上放两个小控制柄，第二个div上放一个控制柄，第三个控制柄放到div外边，控制柄不会随着div一起拖动。这样，只有鼠标放到控制柄的位置上才能拖动对应的div，抓住其他部分是不起作用的。

先看一下div的结构，首先在原来的div标签基础上添加控制柄对应的div内容，如下面的代码所示。

```
<div id="dd-demo-1" class="dd-demo">
    <div id="dd-handle-1a" class="dd-multi-handle-1">H1</div>
    <div id="dd-handle-1b" class="dd-multi-handle-2">H2</div>
</div>
<div id="dd-demo-2" class="dd-demo">
    <div id="dd-handle-2" class="dd-handle">H</div>
</div>
<div id="dd-handle-3" class="dd-outer-handle">Outer</div>
<div id="dd-demo-3" class="dd-demo"></div>
```

在上述代码中，前两个控制柄放在对应div的内部，而第三个控制柄放在对应div的外部，现在我们为这些控制柄添加控制功能，如下面的代码所示。

```
dd1 = new Ext.dd.DD("dd-demo-1");
    dd1.setHandleElId("dd-handle-1a");
    dd1.setHandleElId("dd-handle-1b");
dd2 = new Ext.dd.DD("dd-demo-2");
    dd2.setHandleElId("dd-handle-2");
dd3 = new Ext.dd.DD("dd-demo-3");
    dd3.setOuterHandleElId("dd-handle-3");
```

用法很简单，只要调用Ext.dd.DD的setHandleElId()函数并绑定对应控制柄的id即可。而setOuterHandleElId()函数是专门用来指定外部控制柄的。通过这些配置，我们就限制用户只能通过控制柄对div进行拖放了。不过，外部的控制柄不会跟着被拖动的div一起移动。

示例在06.dnd/04-02.html中。

6.5.3 总在最上面

为了便于拖放，我们希望当前正在拖放的div总显示在最上面（On Top），不会被其他元素

遮挡住，这样才能看清楚到底把div拖放到了什么位置。

为了实现这个效果，我们就要重写监听拖放事件的函数，见代码清单6-2。

代码清单6-2　为OnTop重写监听拖放事件的函数

```
Ext.ux.DDOnTop = function(id, sGroup, config) {
    Ext.ux.DDOnTop.superclass.constructor.apply(this, arguments);
};

Ext.extend(Ext.ux.DDOnTop, Ext.dd.DD, {
    origZ: 0,

    startDrag: function(x, y) {
        var style = this.getEl().style;
        this.origZ = style.zIndex;
        style.zIndex = 999;
    },
    endDrag: function(e) {
        this.getEl().style.zIndex = this.origZ;
    }
});
```

我们在这里定义了一个新对象Ext.ux.DDOnTop，它继承自Ext.dd.DD。我们为Ext.ux.DDOnTop重写了函数startDrag()和endDrag()。这样，在开始拖放对象时会执行startDrag()，把正在拖放的元素对应el的zIndex树形设置成999。这个值已经很大了，基本可以保证当前元素一直显示在所有元素的最上面，待停止拖放时再执行endDrag()，把对应元素el的zIndex树形恢复成原来的数据。

下面不需要再进行修改了，创建3个Ext.ux.DDOnTop对象即可。这样就实现了总在最上面的效果，如下面的代码所示。

```
dd1 = new Ext.ux.DDOnTop("dd-demo-1");
dd2 = new Ext.ux.DDOnTop("dd-demo-2");
dd3 = new Ext.ux.DDOnTop("dd-demo-3");
```

运行结果如图6-9所示。

该示例在06.dnd/04-03.html中。

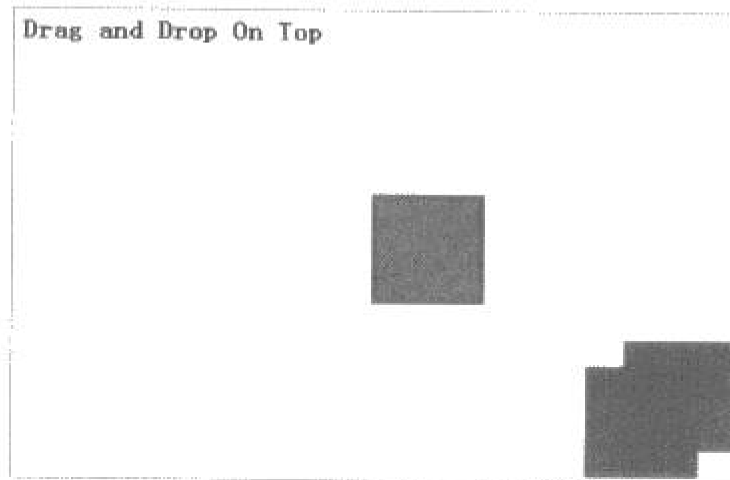


图6-9　总在最上面（On Top）示例效果图

6.5.4 代理

所谓代理（Proxy），就是拖放时原div不动，跟随鼠标移动的是一个名为Proxy的拖放。如果需要拖动的对象十分复杂，每次都让它跟随鼠标移动，那么很可能会使浏览器负荷过大，甚至出现页面停顿的现象。这时就需要使用代理来避免重复渲染复杂对象，代理的效果如图6-10所示。

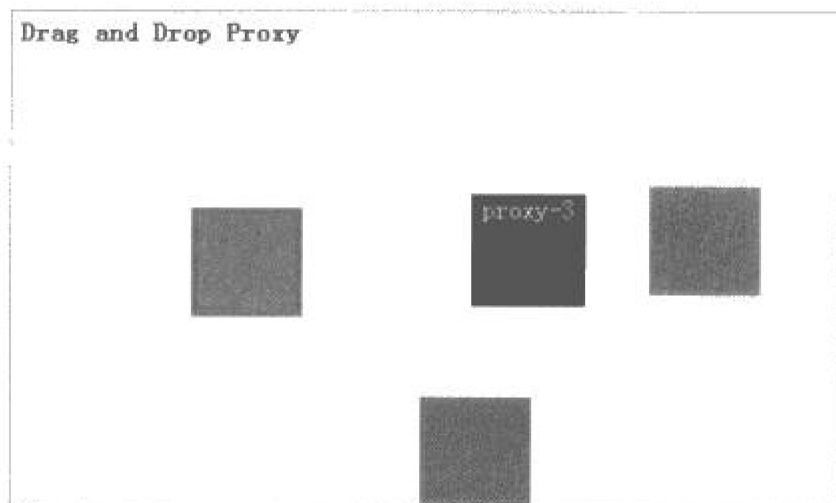


图6-10 代理（Proxy）示例效果图

为了使用代理，最简单的办法就是把原来的Ext.dd.DD改成Ext.dd.DDProxy，如下面的代码所示。

```
dd1 = new Ext.dd.DDProxy("dd-demo-1");
dd2 = new Ext.dd.DDProxy("dd-demo-2");
```

这样就会出现一个只有外框的空白Proxy。如果我们希望自定义Proxy的形式，也可以制作图6-10中演示的黑色Proxy，如下面的代码所示。

```
dd3 = new Ext.dd.DDProxy("dd-demo-3", "default", {
    dragElId: "dd-demo-3-proxy",
    resizeFrame: false
});
```

为了实现自定义Proxy，我们在创建DDProxy时需要设置3个参数。第一个参数是对应的div的id；第二个参数是拖放的组，只有同组的Drag才能放到同组的Drop上（该参数我们现在还不会用到，暂不考虑）；第三个参数是附加参数。

dragElId的值是我们自定义的proxy，而resizeFrame:false告诉EXT不用使proxy的大小与原div一样。

看下面的代码，第三个proxy与dd-demo-3对应，我们还需要在HTML里加上这个代理对应的div，如下面的代码所示。

```
<div id="dd-demo-3-proxy">proxy-3</div>
```

该示例在06.dnd/04-04.html中。

6.5.5 分组

分组（Group）这个示例比较复杂，显示效果如图6-11所示，分组功能的实现是内置在Ext.dd中的，使用起来很方便。

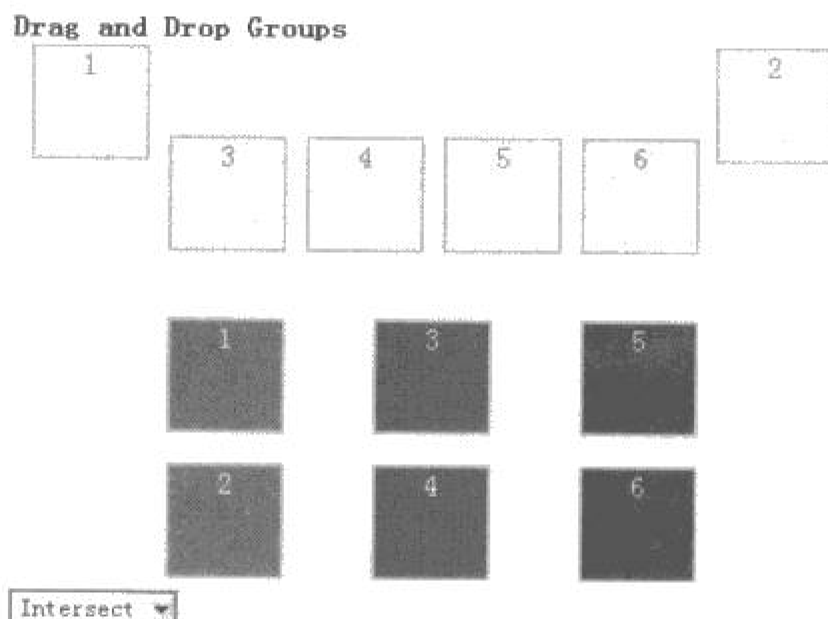


图6-11 分组（Group）示例效果图

简单说明一下，图6-11下方有6个正方形，上方也有6个正方形区域，我们要做的就是把下方的正方形拖到上面的正方形区域中。我们使用拖放分组限制了拖放的方式，下面的6个正方形的拖放形式是有区别的：1号和2号只能放到上边的1号和2号上；3号和4号可以放到上面的3、4、5、6号上；5号和6号可以放到上面任何一个框上。这种方式就叫做分组，只有同一组中的元素才能互相拖放。

在Ext.dd中，可以拖放的元素称为DDProxy，可以让这些DDProxy放下的区域叫做DDTarget。我们可以把DragProxy拖动到同组的DDTarget中，这样就限制了拖放的范围。

HTML中的内容如代码清单6-3所示。

代码清单6-3 分组的页面布局

```
<div id="workarea">
    <div class="slot" id="t1" >1</div>
    <div class="slot" id="t2" >2</div>
    <div class="slot" id="b1" >3</div>
    <div class="slot" id="b2" >4</div>
    <div class="slot" id="b3" >5</div>
    <div class="slot" id="b4" >6</div>
    <div class="player" id="pt1" >1</div>
    <div class="player" id="pt2" >2</div>
    <div class="player" id="pb1" >3</div>
    <div class="player" id="pb2" >4</div>
    <div class="player" id="pboth1" >5</div>
    <div class="player" id="pboth2" >6</div>
</div>
```

```
<select id="ddmode" >
    <option value="0" selected>Point</option>
    <option value="1">Intersect</option>
</select>
```

上面的HTML代码包括了图6-11上方的6个正方形和下方的6个正方形区域，另外还有一个下拉框用来选择拖放碰撞的检测模式。现在来看看如何在JavaScript中将这些HTML元素转换成可拖放的组件。

首先为上方的6个正方形创建DDTarget，如下面的代码所示。

```
var slots = [];
// 槽位
slots[0] = new Ext.dd.DDTarget("t1", "topslots");
slots[1] = new Ext.dd.DDTarget("t2", "topslots");
slots[2] = new Ext.dd.DDTarget("b1", "bottomslots");
slots[3] = new Ext.dd.DDTarget("b2", "bottomslots");
slots[4] = new Ext.dd.DDTarget("b3", "bottomslots");
slots[5] = new Ext.dd.DDTarget("b4", "bottomslots");
```

第一个参数是DDTarget对应的id，第二个参数是组名。这里把6个DDTarget分成了两组topslots和buttomslots，以后的操作就都是基于这两个组的。

接下来就是操作Ext.dd.DDProxy了。为了实现突出显示CSS样式，继承Ext.dd.DDProxy实现了新的类型Ext.ux.DDPlayer，这些细节并不会影响拖放的分组操作。创建分组拖放对象的过程如代码清单6-4所示。

代码清单6-4 创建分组拖放对象

```
var players = [];
// 拖放对象
players[0] = new Ext.ux.DDPlayer("pt1", "topslots");
players[1] = new Ext.ux.DDPlayer("pt2", "topslots");
players[2] = new Ext.ux.DDPlayer("pb1", "bottomslots");
players[3] = new Ext.ux.DDPlayer("pb2", "bottomslots");
players[4] = new Ext.ux.DDPlayer("pboth1", "topslots");
players[4].addToGroup("bottomslots");
players[5] = new Ext.ux.DDPlayer("pboth2", "topslots");
players[5].addToGroup("bottomslots");
```

1号和2号DDPlayer对应的分组是topslots；3号和4号DDPlayer对应的分组是bottomslots；5号和6号DDPlayer对应的分组是topslots，然后调用addToGroup()函数将这两个DDPlayer添加到bottomslots组中。这样一来，5号和6号DDPlayer与两个组都绑定了，它们可以拖放到这两个组中所有的DDTarget上了。

最后的部分是为Ext.dd.DragDropMgr设置碰撞检测模式：POINT和INTERSECT。POINT对应的值是0，INTERSECT对应的值是1，如下面的代码所示。

```
Ext.dd.DragDropMgr.mode = Ext.get('ddmode').dom.selectedIndex;

Ext.get('ddmode').on('change', function() {
    Ext.dd.DragDropMgr.mode = Ext.get('ddmode').dom.selectedIndex;
});
```

这两者的区别是，在POINT模式下，当拖放的鼠标进入DDTarget的范围时才能放下；在INTERSECT模式下，当拖放的DDProxy边缘与DDTarget有重叠时才可以放下。

Ext.ux.DDPlayer的内部很复杂，它继承自Ext.dd.DDProxy，在内部通过Ext.dd.DragDropMgr来操作相互之间有关联的元素。我们使用Ext.ux.DDPlayer实现拖放功能的代码如下所示：

```
Ext.ux.DDPlayer = function(id, sGroup, config) {
    Ext.ux.DDPlayer.superclass.constructor.apply(this, arguments);
    this.initPlayer(id, sGroup, config);
};

Ext.extend(Ext.ux.DDPlayer, Ext.dd.DDProxy, {

    TYPE: "DDPlayer",

    initPlayer: function(id, sGroup, config) {
        if (!id) {
            return;
        }

        var el = this.getDragEl();
        var elem = Ext.get(el);
        elem.setStyle("borderColor", "transparent");
        elem.setStyle("opacity", 0.76);

        this.isTarget = false;

        this.originalStyles = [];

        this.type = Ext.ux.DDPlayer.TYPE;
        this.slot = null;

        this.startPos = Ext.get(this.getEl()).getXY();
    },

    startDrag: function(x, y) {

        var dragEl = this.getDragEl();
        var clickEl = this.getEl();

        dragEl.innerHTML = clickEl.innerHTML;
        dragEl.className = clickEl.className;

        Ext.get(dragEl).setStyle("color", Ext.get(clickEl).getStyle("color"));
        Ext.get(dragEl).setStyle("backgroundColor", Ext.get(clickEl).getStyle
            ("backgroundColor"));
        Ext.get(clickEl).setStyle("opacity", 0.1);

        var targets = Ext.dd.DragDropMgr.getRelated(this, true);
        for (var i=0; i<targets.length; i++) {

            var targetEl = this.getTargetDomRef(targets[i]);

            if (!this.originalStyles[targetEl.id]) {
                this.originalStyles[targetEl.id] = targetEl.className;
```

```
            }

            targetEl.className = "target";
        }
    },

    getTargetDomRef: function(oDD) {
        if (oDD.player) {
            return oDD.player.getEl();
        } else {
            return oDD.getEl();
        }
    },

    endDrag: function(e) {
        Ext.get(this.getEl()).setStyle("opacity", 1);

        this.resetTargets();
    },

    resetTargets: function() {
        var targets = Ext.dd.DragDropMgr.getRelated(this, true);
        for (var i=0; i<targets.length; i++) {
            var targetEl = this.getTargetDomRef(targets[i]);
            var oldStyle = this.originalStyles[targetEl.id];
            if (oldStyle) {
                targetEl.className = oldStyle;
            }
        }
    },

    onDragDrop: function(e, id) {
        var oDD;

        if ("string" == typeof id) {
            oDD = Ext.dd.DragDropMgr.getDDById(id);
        } else {
            oDD = Ext.dd.DragDropMgr.getBestMatch(id);
        }

        var el = this.getEl();

        if (oDD.player) {
            if (this.slot) {
                if ( Ext.dd.DragDropMgr.isLegalTarget(oDD.player, this.slot) ) {
                    Ext.dd.DragDropMgr.moveToEl(oDD.player.getEl(), el);
                    this.slot.player = oDD.player;
                    oDD.player.slot = this.slot;
                } else {
                    Ext.dd.DragDropMgr.setXY(oDD.player.getEl(), oDD.player.startPos);
                    this.slot.player = null;
                    oDD.player.slot = null
                }
            } else {
                oDD.player.slot = null;
                Ext.dd.DragDropMgr.moveToEl(oDD.player.getEl(), el);
            }
```

6

```
            } else {
                if (this.slot) {
                    this.slot.player = null;
                }
            }

            Ext.dd.DragDropMgr.moveToEl(el, oDD.getEl());
            this.resetTargets();

            this.slot = oDD;
            this.slot.player = this;
        },

        swap: function(el1, el2) {
            var pos1 = Ext.get(el1).getXY();
            var pos2 = Ext.get(el2).getXY();

            Ext.get(el1).setXY(pos2);
            Ext.get(el2).setXY(pos1);
        },

        onDragOver: function(e, id) {
        },

        onDrag: function(e, id) {
        }

    });
```

该示例在06.dnd/04-05.html中。

6.5.6 网格

网格（Grid）为拖放对象设置步长，每次拖放时不再平滑移动，而是沿网格非线性移动（见图6-12）。如果可以根据这个功能自定义页面模板布局，应该是非常不错的。

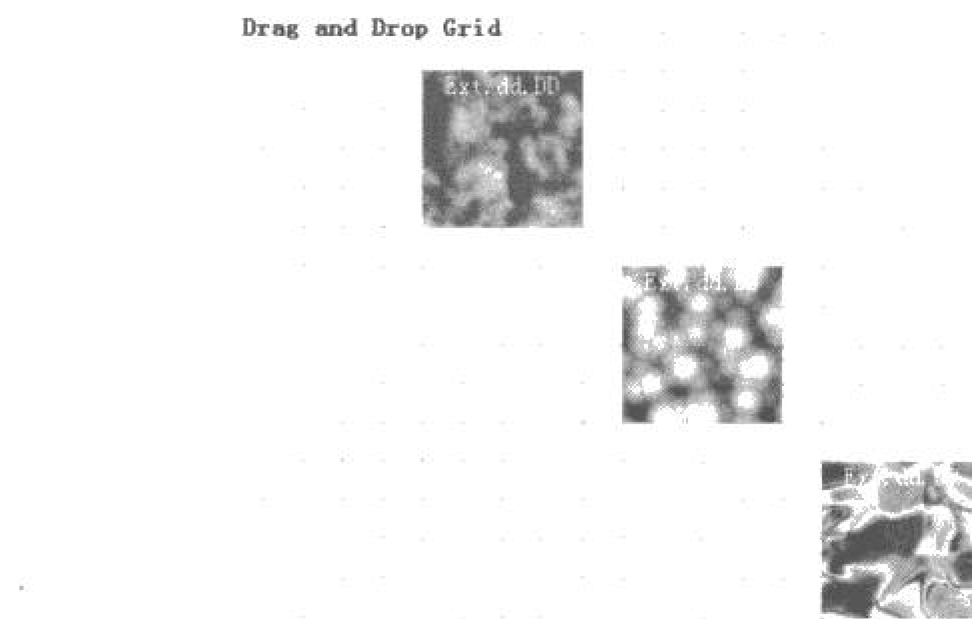


图6-12 网格（Grid）示例效果图

这里是HTML中的定义，如下面的代码所示。

```
<div id="dragDiv1" class="testSquare">Ext.dd.DD</div>
<div id="dragDiv2" class="testSquare">Ext.dd.DD</div>
<div id="dragDiv3" class="testSquare">Ext.dd.DD</div>
```

页面上显示的网格和拖放的图片都是通过CSS定义的，背景中使用的定义如下面的代码所示。

```
body { background: url("img/grid.png") }
```

有了这些准备，我们再利用Ext.dd来制作拖放对象。此外，还要对拖放对象的移动距离进行控制，如下面的代码所示。

```
dd1 = new Ext.dd.DD("dragDiv1");
dd1.setXConstraint(1000, 1000, 25);
dd1.setYConstraint(1000, 1000, 25);
dd2 = new Ext.dd.DD("dragDiv2");
dd2.setXConstraint(1000, 1000, 25);
dd2.setYConstraint(1000, 1000, 25);
dd3 = new Ext.dd.DD("dragDiv3");
dd3.setXConstraint(1000, 1000, 25);
dd3.setYConstraint(1000, 1000, 25);
```

生成Ext.dd.DD对象的形式与之前完全一样，不同的是为这些拖放对象设置移动距离。注意函数setXConstraint()和setYConstraint()，这两个函数各有3个参数。setXConstraint()的3个参数分别是左侧可以达到的最远距离（以像素为单位），右侧可以达到的最远距离和每次移动的距离；setYConstraint()的3个参数分别是上面可以达到的最远距离，下面可以达到的最远距离和每次移动的距离。这样，左、右和上、下可移动的范围都是从–1000到1000，每次移动25像素，正好与背景方格的大小一样。

该示例在06.dnd/04-06.html中。

6.5.7 拖动圆形

当然，在HTML里是不能画圆（Circle）的，我们使用的是一个圆形图片。为了让效果更逼真，在拖放时，检测碰撞的算法还是把这个拖放体当作圆形来看待。接下来，我们可以讨论一下如何自定义碰撞边界。

图6-13中的圆形就是可拖放对象，正方形是与其对应的DDTarget，只有把圆形拖到正方形上才能放下，否则又会弹回原地。

Drag and Drop Circle





图6-13 圆形（Circle）示例效果图

为了实现将圆形放到方形区域上的效果，我们需要判断圆形与正方形区域叠放在一起的时机，需要知道什么时候圆形与方形区域出现了重叠，然后才能执行将圆形放入方形区域的操作，具体步骤如下所示。

(1) 获得拖放对象的初始位置，如下面的代码所示。

```
var clickRadius = 46;
```

```
var el = Ext.get("dd-demo-1");
startPos = el.getXY();
```

clickRadius是圆形的半径，用来计算碰撞边界。

(2) 生成拖放对象，如下面的代码所示。

```
dd = new Ext.dd.DD(el);
```

(3) 为拖放对象设置校验函数（见代码清单6-5）。

代码清单6-5 拖放校验函数

```
dd.clickValidator = function(e) {

    var el = this.getEl();
    var region = Ext.get(el).getRegion();
    var r = clickRadius;
    var x1 = e.getPageX(), y1 = e.getPageY();
    var x2 = Math.round((region.right + region.left) / 2);
    var y2 = Math.round((region.top + region.bottom) / 2);

    e.preventDefault();

    return ( ((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2)) <= r*r );
};
```

首先获得DragTarget中的region，其中保存了上、下、左、右4个边界。用它可以计算出DragTarget中心点的位置（x2,y2），然后再获得（x1,y1），这是拖放过程中鼠标的*x*坐标和*y*坐标。

然后计算一下（x1,y1）和（x2,y2）这两点之间的距离，也就是鼠标与DragTarget之间的距离。如果这个距离小于半径，说明圆形可以放到DragTarget上，否则就是无效的。在这两种情况下会分别执行函数onDragDrop()和onInvalidDrop()。

onDragDrop()会把拖放对象的位置设置成DragTarget的位置，看起来就像圆形放到了DragTarget中间一样，如下面的代码所示。

```
dd.onDragDrop = function(e, id) {
    Ext.get(this.getEl()).setXY(Ext.get(id).getXY());
}
```

onValidDrop()会让拖放对象返回原位，moveTo()函数的前两个坐标是*x*和*y*，第三个参数true表示开启动画效果，如下面的代码所示。

```
dd.onInvalidDrop = function(e) {
    Ext.get(this.id).moveTo(startPos[0], startPos[1], true);
}
```

(4) 设置DDTarget，如下面的代码所示。

```
dd2 = new Ext.dd.DDTarget("dd-demo-2");
```

所有步骤都已经完成了，现在可以尝试拖放圆形了。

该示例在06.dnd/04-07.html中。

6.5.8 拖动范围

在页面上画一个矩形，让拖放对象只能在这个矩形里任意拖动，这就是限制拖放的范围（Region），如图6-14所示。

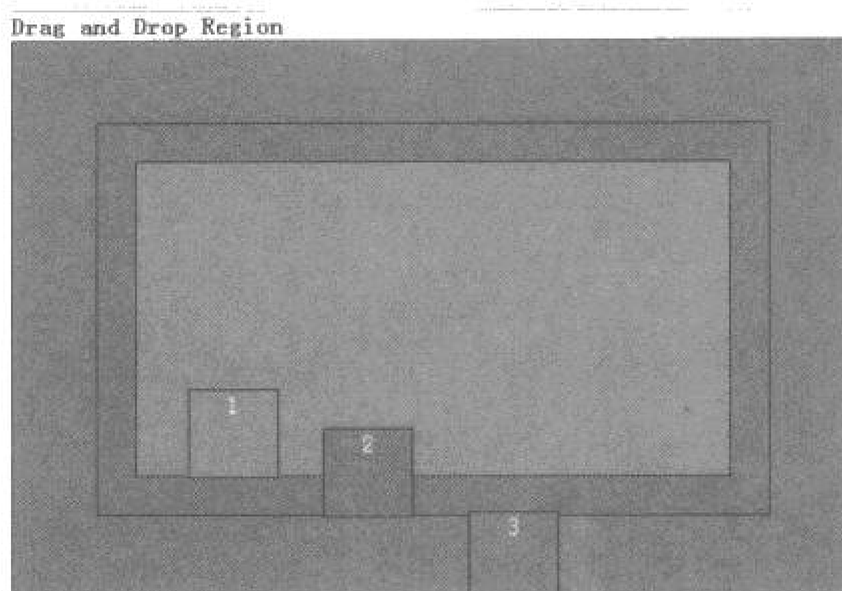


图6-14 拖动范围（Region）示例效果图

如图6-14所示，蓝色矩形（1号）只能在蓝色区域里移动，绿色矩形（2号）只能在绿色方框里移动，棕色矩形（3号）只能在棕色区域里移动。

看一下示例的HTML内容，把3个可拖放的div放到3个不同大小的div里，如代码清单6-6所示。

代码清单6-6 示例的HTML内容

```
<div id="dd-demo-canvas1">
    <div id="dd-demo-canvas2">
        <div id="dd-demo-canvas3">
            <div id="dd-demo-1" class="dd-demo"><div>1</div></div>
            <div id="dd-demo-2" class="dd-demo"><div>2</div></div>
            <div id="dd-demo-3" class="dd-demo"><div>3</div></div>
        </div>
    </div>
</div>
```

dd-demo-1、dd-demo-2、dd-demo-3是可拖放的对象，dd-demo-canvas1、dd-demo-canvas2、dd-demo-canvas3对应的是拖放范围。为了实现该效果，还专门创建了Ext.ux.DDRegion对象，如下面的代码所示。

```
dd1 = new Ext.ux.DDRegion('dd-demo-1', '', { cont: 'dd-demo-canvas3' });
dd2 = new Ext.ux.DDRegion('dd-demo-2', '', { cont: 'dd-demo-canvas2' });
dd3 = new Ext.ux.DDRegion('dd-demo-3', '', { cont: 'dd-demo-canvas1' });
```

DDRegion继承自Ext.dd.DD，所以创建过程中的3个参数分别是：拖放div的id、拖放组和附加参数。这里的附加参数中包含的cont对应了可拖放区域的id，这个参数是设置在DDRegion中的。让我们回头看看在DDRegion中是如何做的吧，如下面的代码所示。

```
Ext.ux.DDRegion = function(id, sGroup, config) {
    this.cont = config.cont;
    Ext.ux.DDRegion.superclass.constructor.apply(this, arguments);
};
```

首先在构造函数里把config.cont赋给this.cont，然后调用DDRegion父类的构造函数进行初始化，如代码清单6-7所示。

代码清单6-7　拖放范围

```
Ext.extend(Ext.ux.DDRegion, Ext.dd.DD, {
    cont: null,
    init: function() {
        Ext.ux.DDRegion.superclass.init.apply(this, arguments);
        this.initConstraints();
    },
    initConstraints: function() {
        var region = Ext.get(this.cont).getRegion();

        var el = this.getEl();
        var xy = Ext.get(el).getXY();

        var width = parseInt(Ext.get(el).getStyle('width'), 10);
        var height = parseInt(Ext.get(el).getStyle('height'), 10);

        var left = xy[0] - region.left;
        var right = region.right - xy[0] - width;

        var top = xy[1] - region.top;
        var bottom = region.bottom - xy[1] - height;

        this.setXConstraint(left, right);
        this.setYConstraint(top, bottom);
    }
});
```

剩下的这些都是在initConstraints()函数里进行配置的。this.cont计算出它所在的region，获得的是可拖放区域的上、下、左、右4个边界，然后把这4个值通过setXConstraint()和setYConstraint()两个函数设置给Ext.dd.DD，这样就限定了4个方向上的可移动范围。说到底，其实还是利用了setXConstaint()和setYConstraint()两个函数定义拖放移动的范围。

该示例在06.dnd/04-08.html中。

6.6　小结

本章讲解了EXT中与拖放有关的知识，介绍了最基本的拖放方式和EXT中拖放功能的简单应用，然后展示了EXT中的拖放组件关系图，并介绍了3大类拖放事件流程。

最后以YUI中的拖放示例为基础，讲解了8个典型示例：基础（Basic）、控制柄（Handle）、总在最上面（On Top）、代理（Proxy）、分组（Group）、网格（Grid）、拖动圆形（Circle）、拖动范围（Region），从而让我们了解拖放的高级配置。

第7章 弹出窗口

本章内容

- Ext.MessageBox
- 对话框的更多配置
- Ext.window的常用属性
- 窗口分组
- 向窗口中放入各种控件

7.1 Ext.MessageBox

从外观上来讲，浏览器自带的alert、confirm、prompt等对话框并不好看，而且配置也不灵活。诸如按钮的添加、删除，以及修改按下按钮所触发的事件等操作都非常难以执行。

幸运的是，这些功能在EXT的msgbox里都能实现，而且外观也很漂亮。实际上，EXT中常用到的MessageBox也是一种特殊的窗口（window）。窗口的配置很简单，可以任意拖动，组件可以任意摆放，可以使用任何控件。如果使用tab进行切换，那么还能实现窗口的最大化。

Ext.MessageBox为我们提供了弹出提示框的简单方法，使用它提供的alert、confirm、prompt等对话框完全可以替代浏览器原生的alert、confirm、prompt等对话框。除此之外，Ext.MessageBox还提供了诸如进度条等更多的功能。

7.1.1 Ext.MessageBox.alert()

下面我们先看一下消息对话框的alert对话框示例，代码如下所示。

```
Ext.MessageBox.alert('标题', '内容', function(btn) {
    alert('你刚刚点击了 ' + btn);
});
```

代码运行的结果如图7-1所示。



图7-1 alert对话框

第一个参数可用于修改窗口的标题，第二个参数用于决定窗口的内容，第三个参数是关闭按钮之后（无论是单击OK按钮，还是单击右上角的关闭按钮）就会执行的函数，即回调函数。

在此，我们解释一下回调函数的作用。以前使用过JavaScript中的alert的读者都会知道，弹出

系统原生的alert对话框之后，整个脚本都会变成阻塞状态，在用户关闭alert对话框之前脚本不会继续向下执行。这种阻塞效果是浏览器特有的功能，在JavaScript中无法模仿这种效果。但是，在实际开发中确实需要知道alert对话框在何时被关闭了，并且需要在alert对话框关闭后执行一些操作。为了保证这种先后的执行顺序，`Ext.MessageBox`为用户提供了回调函数，作为`Ext.MessageBox.alert()`的第三个参数，这个回调函数会在alert对话框关闭时执行。

该示例在07.window/01.html中。

7.1.2 Ext.MessageBox.confirm()

confim对话框为用户提供了两个选项：Yes或No，它们会在需要用户做出判断时用到。最常见的情况是，当用户选择删除某个数据时，系统会弹出对话框询问用户是否确认删除操作，这样可以避免用户因为误操作而删除数据，如下面的代码所示。

```
Ext.MessageBox.confirm('选择框', '你到底是选择Yes还是No? ', function(btn) {
    alert('你刚刚点击了 ' + btn);
});
```

运行结果如图7-2所示。

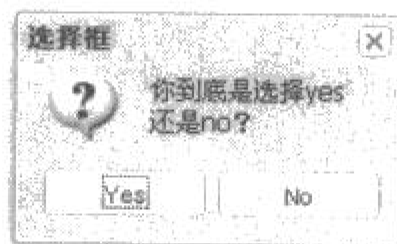


图7-2　confirm对话框

因为confirm对话框需要在用户选择了某个选项时进行相应的处理，所以回调函数是必不可少的。

在用户选择Yes或No之后，回调函数即被调用，`btn`参数的值可能是`yes`或`no`，通过它可以知道用户点击了哪个按钮。

该示例在07.window/01.html中。

7.1.3 Ext.MessageBox.prompt()

prompt对话框为用户提供了更大的灵活性，允许用户输入一段字符串，然后提交给JavaScript处理。它还提供了两个按钮，用户可以决定是将输入的内容提交给JavaScript，还是取消这次输入操作，如下面的示例所示。

```
Ext.MessageBox.prompt('输入框', '随便输入一些东西', function(btn, text) {
    alert('你刚刚点击了 ' + btn + ', 刚刚输入了 ' + text);
});
```

运行结果如图7-3所示。



图7-3　prompt对话框

在此，我们先在文本框中输入一些文字，然后单击任何一个按钮，就会立刻弹出信息提示，提示的内容为刚才单击的按钮和输入的文字。

`prompt`的回调函数有两个参数：第一个参数`btn`代表用户单击的按钮，可能是OK或Cancel；第二个参数`text`就是用户刚才在文本框中输入的内容。

该示例在07.window/01.html中。

7.2 对话框的更多配置

以上我们讨论了用于替换浏览器原生的alert、confirm和prompt的3个函数。下面来看一下Ext.MessageBox提供的其他功能。

7.2.1 可以输入多行的输入框

修改原来的prompt函数，将原来只能接收单行字符串数据的文本框修改为可以支持多行文字的形式，如图7-4所示。



图7-4 多行输入

从图7-4中可以看出，原来的textfield现在已经变成了textarea。不仅如此，我们还修改了对话框的标题和提示内容。这些都是通过Ext.MessageBox.show()来实现的，它接受一个JSON对象作为参数，相关代码如下所示。

```
Ext.MessageBox.show({
    title: '多行输入框',
    msg: '你可以输入好几行',
    width:300,
    buttons: Ext.MessageBox.OKCANCEL,
    multiline: true,
    fn: function(btn, text) {
        alert('你刚刚点击了 ' + btn + '，刚刚输入了 ' + text);
    }
});
```

注意Ext.MessageBox.show()中用到的参数：

- Title表示弹出对话框的标题；
- Msg表示在弹出对话框中显示的提示内容；
- Buttons表示对话框中的按钮，Ext.MessageBox已经为我们预设好了按钮设置。这里使用的是Ext.MessageBox.OKCANCEL，即显示OK和Cancel两个按钮；
- Multiline表示可以输入多行内容，设置为true之后便自动生成一个textarea供我们使用；
- 最后的fn表示回调函数，它会在用户关闭对话框时接收并处理我们需要的结果。

7.2.2 自定义对话框的按钮

我们还可以使用Ext.Message.show()设置对话框中按钮的样式，如下面的代码所示。

```
Ext.MessageBox.show({
    title:'随便按个按钮',
    msg: '从三个按钮里随便选择一个',
    buttons: Ext.MessageBox.YESNOCANCEL,
    fn: function(btn) {
        alert('你刚刚点击了 ' + btn);
    }
});
```

运行结果如图7-5所示。

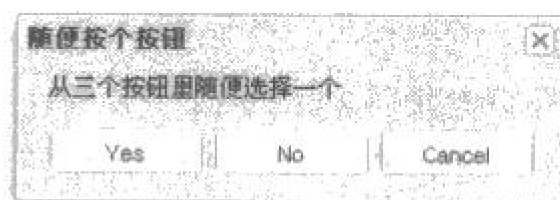


图7-5 自定义按钮

这次我们依然设置了title（标题）、msg（提示内容）、buttons（按钮）和fn（回调函数）等参数。

我们为buttons参数设置了Ext.MessageBox.YESNOCANCEL，这样可以在弹出的对话框中看到3个按钮。YESNOCANCEL是定义在EXT中的一个变量，它的值是{yes:true, no:true, cancel:true}，Ext.MessageBox中预设的4个按钮分别是OK、Yes、No和Cancel。如果不使用YESNOCANCEL这种预设变量，也可以直接使用{ok:true,yes:true,no:true,cancel:true}的形式，这样4个按钮都会显示在对话框中。

7.2.3 进度条

进度条经常用于需要用户等待某一操作完成的场景。在执行一些十分耗时的操作时，我们需要用它来提示用户耐心等待。Ext.MessageBox为我们提供了默认的进度条，只要将progress参数设置为true，对话框中就会出现进度条，如下面的代码所示。

```
Ext.MessageBox.show({
    title: '请等待',
    msg: '读取数据中',
    width:240,
    progress:true,
    closable:false
});
```

也可以直接使用Ext.MessageBox提供的progress函数，如下面的代码所示。

```
Ext.MessageBox.progress('请等待', msg: '读取数据中');
```

其运行结果如图7-6所示。



图7-6 进度条

我们已经得到了进度条，但是它的进度状态不会发生变化（自动向前推进），需要调用Ext.MessageBox.updateProgress()函数让进度条状态发生变化。通常，我们会使用closable: false来隐藏对话框右上角的关闭按钮，从而禁止用户关掉进度条。

现在，为上例添加更新进度条的功能，我们使用timeout定时器对进度条进行修改。进度条的状态会随着时间（以秒为单位）而变化，10秒之后整个进度条对话框将隐藏，如下面的代码所示。

```
var f = function(v){
    return function(){
        if(v == 11){
            Ext.MessageBox.hide();
        }else{
            Ext.MessageBox.updateProgress(v/10, '正在读取第 ' + v + ' 个，一共10个。');
        }
    };
```

```
};
for(var i = 1; i < 12; i++){
    setTimeout(f(i), i*1000);
}
```

除了这种可以精确控制进度的进度条，我们还可以使用一种自动变化的进度条提示框，这时要使用Ext.Message.wait()。

```
Ext.MessageBox.wait('请等待', msg: '读取数据中');
```

这时弹出的对话框中的进度条就会自动向前推进了，不过我们无法对进度条的值进行精确控制。如果等待操作的时间过长，进度条满格之后又会从零开始向前推进。请根据具体情况选择使用Ext.MessageBox.progress()或Ext.MessageBox.wait()。

7.2.4 动画效果

我们可以为对话框设置弹出和关闭的动画效果。使用animEl参数指定一个HTML元素，对话框就会依据指定的HTML元素播放弹出和关闭的动画。我们把前面的示例稍作修改，加入animEl参数实现动画效果，如下面的代码所示。

```
Ext.MessageBox.show({
    title:'随便按个按钮',
    msg: '从三个按钮里随便选择一个',
    buttons: Ext.MessageBox.YESNOCANCEL,
    fn: function(btn) {
        alert('你刚刚点击了 ' + btn);
    },
    animEl: 'dialog'
});
```

animEl参数的值是一个字符串，它与HTML中的一个元素的id相对应，例如<div id="dialog"></div>。依据这个元素的id，我们创建的对话框才知道根据哪个元素播放弹出和关闭的动画。

示例在07.window\01.html中。

除了Ext.MessageBox的基本应用，实际使用时还需要注意以下两点。

- 为了简化调用，可以把Ext.MessageBox直接写成Ext.Msg，如Ext.Msg.alert()、Ext.Msg.confirm()、Ext.Msg.prompt()等。
- 我们使用Ext.MessageBox弹出的对话框都是基于同一个内部Ext.Window实例的，因此不能使用Ext.MessageBox弹出多个对话框，这样操作的结果是页面上只会显示最后一个窗口。这个问题在使用时要特别注意。

7.3 Ext.window 的常用属性

下面来具体讨论一下Window的基本属性和应用。从外表来看，Window和MessageBox在外形方面的功能很相似，因为Ext.MessageBox内部也是基于Ext.Window实现的。

7.3.1 创建窗口

下面直接使用Window创建自定义窗口。先看一个简单的示例，如下面的代码所示。

```
var win = new Ext.Window({
    el:'window-win',
    layout:'fit',
    width:500,
    height:300,
    closeAction:'hide',

    items: [{}],

    buttons: [{
        text:'按钮'
    }]
});
win.show();
```

首先要注意的是，上面的Window需要一个对应的HTML元素。我们用el参数指定了'window-win'，这对应HTML中的<div id="window-win"></div>。我们通过width和height参数设置宽度和高度，以此来确定窗口的整体大小。

其次，我们需要设置的是窗口的布局类型，layout:'fit'表示布局会充满整个窗口。在窗口改变大小时，内部组件也会进行相应的调整。而items参数部分指定的是窗口中放置的子组件，比如表格、树、表单等。

closeAction:'hide'是一个常用参数，它用来控制用户单击右上角的关闭图标后会执行的操作。参数值'hide'表示关闭窗口时执行隐藏命令，之后还可以使用show()函数显示刚刚关闭的窗口。如果使用默认的'close'，会在关闭窗口时把窗口对象销毁，那么就不能使用show()函数来显示窗口了。

如果不想让用户使用右上角的关闭按钮关闭窗口，可以设置closable: false。如果不想让用户随意拖动窗口的位置，可以设置draggable: true。

Buttons参数用于设置显示在窗口底端的按钮。我们演示了窗口中显示一个按钮的效果，但是没有为这个按钮添加事件。

现在可以调用show()，让窗口显示出来，效果如图7-7所示。

图7-7中部的空白对应着参数items:[{}]。如果不为items中的子组件指定xtype，那么默认情况下会生成Ext.Panel类型的对象，因为我们没有为items中设置树形，所以显示出来就是空白的一块。因为使用了FitLayout布局方式，所以窗口内部的Ext.Panel也会根据窗口的大小自动调整自身的大小。



图7-7　简单窗口

该示例在07.window\03-01.html中。

Ext.Window直接继承自Ext.Panel，可以使用Ext.Panel中定义的参数对窗口进行设置，包括标题、上方和下方的工具条，以及内容的折叠展开等功能。

7.3.2 窗口的最大化和最小化

首先，为窗口设置参数maximizable: true，这样窗口的右上角会出现表示最大化的按钮，如图7-8所示。



图7-8 最大化按钮

单击最大化按钮后，窗口自动扩展以充满整个浏览器窗口，并且右上角的最大化按钮会变成恢复原来大小的按钮。

此时，最大化后的窗口还会根据浏览器大小而自动改变，它会随时保持完全充满浏览器窗口。

单击恢复按钮，窗口就会恢复到最大化前的大小和位置，如图7-9所示。

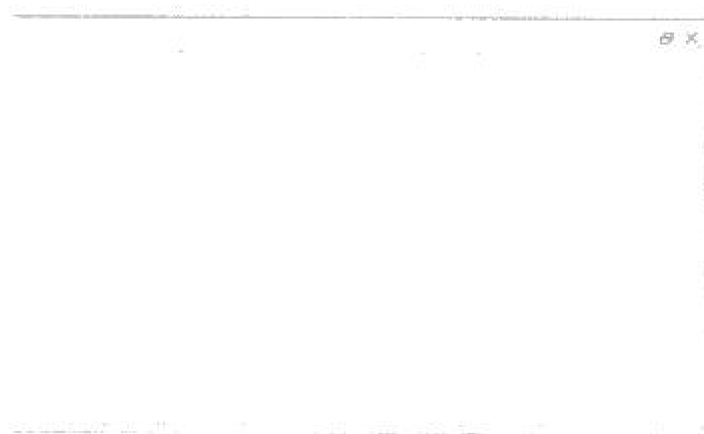


图7-9 恢复窗口大小

最小化窗口的操作与之类似，首先为窗口设置参数minimizable: true，窗口的右上角会出现表示最小化的功能按钮，如图7-10所示。



图7-10 最小化按钮

与最大化窗口不同的是，单击最小化按钮并不会触发任何操作，EXT并没有为窗口预设最小化时的操作，我们需要根据实际情况来自定义最小化功能。有两种实现方法：监听minimize事件和重写minimize()函数。

示例分别在07.window/03-02-01.html和07.window/03-02-02.html中。

7.3.3　窗口的隐藏与销毁

Ext.Window中包含一个closeAction参数，这个参数用来设置在关闭窗口时执行的操作，可以选择隐藏或销毁，默认值为closeAction:'close'。如果使用了closeAction:'close'，那么当用户单击关闭按钮时，EXT就会将窗口对象和其对应的DOM模型完全销毁，销毁了的窗口无法通过调用show()函数显示到页面上。

常见的情况是，我们创建的窗口可能会反复使用多次。用户希望关闭窗口后，还可以通过某种途径再次显示这个窗口。虽然我们可以每次都重新创建同一类型的窗口，但反复的创建和销毁操作会使效率过于低下。常用的方式是在用户单击关闭按钮后并不销毁窗口，而是将窗口隐藏。当用户需要再次使用这个窗口时，会将先前隐藏的窗口显示出来，这样避免了重复的创建与销毁操作，大大提高了程序的效率。

EXT为这种情况提供了对应的参数，我们只需要将closeAction的参数值设置为'hide'就可以实现上述功能。设置过 closeAction:'hide'的窗口在用户单击关闭按钮时就会自动隐藏，之后可以调用show()函数将它显示出来。

如果不愿意借助EXT的内置功能，而是希望自己编写函数控制窗口的隐藏与显示，那么我们也可以使用Ext.Window提供的函数hide()和show()。hide()函数用于隐藏窗口，show()函数用于显示窗口。与closeAction:'close'对应的是close()函数，执行close()函数就会销毁对应窗口，销毁后的窗口也不能再次显示了。

与窗口关闭相关的配置还有closable参数，它的默认值为true。默认情况下，窗口都会在右上角显示关闭按钮，如果设置为closable:false，就会隐藏关闭按钮，效果如图7-11所示。

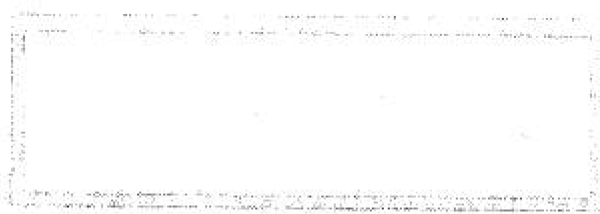


图7-11　隐藏关闭按钮

既然关闭按钮已经被隐藏，我们就无法使用closeAction来配置关闭窗口的操作了。这时就需要采用其他方法，通过手工调用函数close()或hide()来实现关闭窗口的功能。

既然是手动调用，我们需要首先定义一个窗口，代码如下所示：

```
win = new Ext.Window({
        el:'window-win',
        width:300,
        height:100,
        closable: false
    });
win.show();
```

然后在HTML中定义3个按钮，如下面代码所示：

```
<button onclick="win.show()">win.show()</button>
<button onclick="win.hide()">win.hide()</button>
<button onclick="win.close()">win.close()</button>
```

点击这3个按钮看看效果。

示例在07.window/03-03.html中。

7.3.4 防止窗口超出浏览器

默认配置下，窗口可以移动到任何位置，甚至可能超出浏览器的边界。这样有时会导致一些问题，例如，当窗口的位置过于靠近浏览器的上边界时，会遮住顶部的关闭按钮，如图7-12所示。

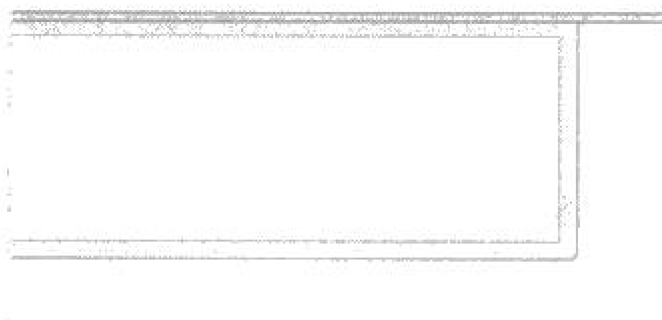


图7-12 窗口超出浏览器

为了避免这种问题，我们应该限制窗口的移动范围，让它始终保持在浏览器的可视范围内。这时可以选择使用参数constrain和constrainHeader，它们分别用来限制窗口的整体和窗口的顶部不能超越浏览器的边界。

使用constrain:true的情况如下面的代码所示。

```
win = new Ext.Window({
    el:'window-win',
    width:300,
    height:100,
    title: 'header',
    html: 'body',
    constrain: true
});
```

运行结果如图7-13所示。

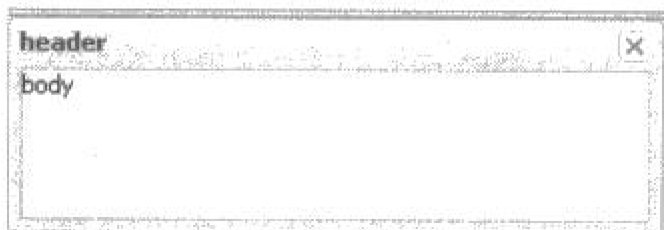


图7-13 设置整个窗口不超出浏览器边界

constrain保证整个窗口不会越过浏览器的边界，完整地显示在浏览器中。

使用constrainHeader:true的情况如下面的代码所示。

```
win = new Ext.Window({
```

```
        el:'window-win',
        width:300,
        height:100,
        title: 'header',
        html: 'body',
        constrainHeader: true
    });
```

运行结果如图7-14所示。



图7-14　只设置窗口顶部不超出浏览器边界

constrainHeader只保证窗口的顶部都不会越过浏览器的显示边界，而窗口的内容部分可以超出浏览器的边界。

上面的两个示例分别在07.window/03-04-02.html和07.window/03-04-03.html中。

7.3.5　设置窗口中的按钮

可以通过buttons参数指定窗口下部的按钮，如代码清单7-1所示。

代码清单7-1　指定窗口下部的按钮

```
var win = new Ext.Window({
    el:'window-win',
    width:300,
    height:100,
    closeAction:'hide',

    buttons: [{
        text: '确定'
    }, {
        text: '取消'
    }]
});
```

运行结果如图7-15所示。

这里设置了“确定”和“取消”两个按钮。默认情况下，生成的按钮会以右对齐的方式排列。如果我们希望这些按钮居中对齐（见图7-16），可以设置buttonAlign: 'center'参数。



图7-15　窗口下部的按钮



图7-16　按钮居中对齐

buttonAlign可以使用的参数有3个：left、center和right，分别代表左对齐、居中对齐和右对齐。如果没有设置buttonAlign，则默认使用右对齐的方式对按钮进行排列。

该示例在07.window/03-05-02.html中。

与按钮相关的设置还有defaultButton参数，它用于在窗口弹出后，默认为用户选择一个按钮（见图7-17）。

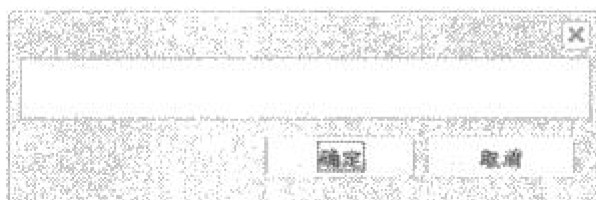


图7-17 设置默认选中的按钮

上例中，我们使用了defaultButton:0，将第一个按钮设置为默认选中状态。如果此时按下键盘上的回车键，就会执行单击“确定”按钮后的操作。“确定”按钮对应的操作是win.hide()，它会隐藏当前窗口，也就是说只需要按下回车键就可以关闭这个窗口，无需再用鼠标单击“确定”按钮。代码如下所示：

```
var win = new Ext.Window({
    el:'window-win',
    width:300,
    height:100,
    closeAction:'hide',
    defaultButton: 0,

    buttons: [{
        text: '确定',
        handler: function() {
            win.hide();
        }
    }, {
        text: '取消'
    }]
});
win.show();
```

7.3.6 窗口的其他配置选项

我们可以使用resizable控制窗口是否可以通过拖放改变大小。在设置resizable:true时，还可以为窗口设置对应的resizeHandles，从而控制拖放的方式（见图7-18）。代码如下所示：

```
var win = new Ext.Window({
        el:'window-win',
        width:300,
        height:100,
        closeAction:'hide',
        resizable: true,
        resizeHandles: 'se'
    });
 win.show();
```

在某些情况下，我们希望弹出窗口之后立即屏蔽掉页面上所有的其他组件，只有在窗口关闭后才能继续操作其他组件，这时可以使用modal:true参数。EXT会为设置了modal:true参数的窗口自动生成一个半透明的DIV，用这个DIV将整个页面遮罩起来，以此模拟使整个页面变灰失效的效果，如图7-19所示。



图7-18　拖放改变窗口大小



图7-19　模拟页面变灰失效的效果

如果你希望使窗口展示弹出并缩回效果的动画，可以用animateTarget指定弹出并缩回效果的HTML元素，如下面的代码所示。

```
var win = new Ext.Window({
    el:'window-win',
    width:300,
    height:100,
    closeAction:'hide',
    animateTarget: 'target'
});
```

还可以使用plain:true参数对窗口内容部分进行美化，如图7-20所示，可以看到整齐的边框。

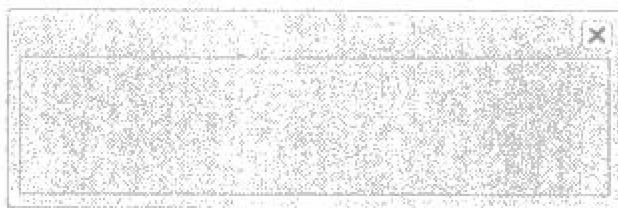


图7-20　使用plain修饰后窗口

该示例在07.window/03-06-04.html中。

7.4　窗口分组

在EXT中，窗口是分组进行管理的，我们可以对某一组中的窗口执行特定的操作。默认情况下，窗口都在Ext.WindowMgr组中。窗口分组由Ext.WindowGroup类定义，该类包括bringToFront、getActive、hideAll、sendToBack等函数，可以用来操作分组中的窗口。窗口分组的实现过程如代码清单7-2所示。

代码清单7-2　窗口分组

```
var i = 0, mygroup;
function newWin() {
    var win = new Ext.Window({title:"窗口"+i++,
```

```
            width: 400,
            height: 300,
            maximizable: true,
            manager: mygroup
        });
        win.show();
    }

    function toBack() {
        mygroup.sendToBack(mygroup.getActive());
    }

    function hideAll() {
        mygroup.hideAll();
    }

    Ext.onReady(function(){
        mygroup = new Ext.WindowGroup();

        Ext.get("btn").on("click",newWin);
        Ext.get("btnToBack").on("click",toBack);
        Ext.get("btnHide").on("click",hideAll);
    });
```

页面中对应的HTML部分如下面的代码所示。

```
<input id="btn" type="button" name="add" value="新窗口" />
<input id="btnToBack" type="button" name="add" value="放到后台" />
<input id="btnHide" type="button" name="add" value="隐藏所有" />
```

执行上面的代码，可以先通过“新窗口”按钮新建几个窗口。因为没有设置窗口弹出的坐标，所以生成的这些窗口会重叠在一起。我们需要拖动这些窗口，把它们排列在屏幕中不同的位置。然后单击“放到后台”按钮，可以把某一个窗口移到该组窗口的最后面。单击“隐藏所有”按钮，可以隐藏当前打开的所有窗口，如图7-21所示。

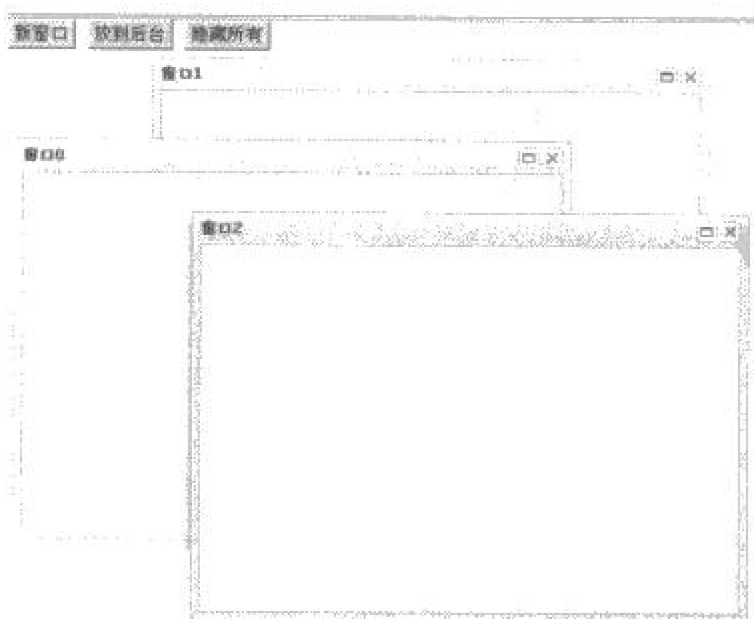


图7-21 窗口组

该示例在07.window\04.html中。

7.5 向窗口中放入各种控件

在EXT中，窗口（Window）继承自Ext.Panel，因此它也支持在内部嵌套其他组件，实现各种复杂的布局。

7.5.1 在窗口中加入表格

首先，创建一个表格。我们这里直接使用第3章的示例，唯一的区别是创建表格之后不必使用render()函数进行渲染，而是将它加入窗口的items参数中，让窗口负责对表格进行布局，效果如图7-22所示。

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3
4	name4	descn4
5	name5	descn5

按钮

图7-22 窗口中的表格

图7-22显示的是一个最基本的表格，我们还可以根据第3章中介绍的功能为它设置分页工具条、排序、右键菜单等功能。实际上，我们只需要单独创建一个表格，配置好需要的功能，然后将它放入窗口中就可以了。

现在我们来看一看在窗口中使用表格时需要注意的一些问题。

- 表格不再需要调用render()，窗口会在显示时自动渲染内部的组件。
- 在窗口中设置表格时，需要把表格加入到窗口的items参数中，如下面代码所示。

```
var cm = new Ext.grid.ColumnModel([
        {header:'编号',dataIndex:'id'},
        {header:'名称',dataIndex:'name'},
        {header:'描述',dataIndex:'descn'}
    ]);

    var data = [
        ['1','name1','descn1'],
        ['2','name2','descn2'],
        ['3','name3','descn3'],
        ['4','name4','descn4'],
        ['5','name5','descn5']
    ];
```

```
        var ds = new Ext.data.Store({
            proxy: new Ext.data.MemoryProxy(data),
            reader: new Ext.data.ArrayReader({}, [
                {name: 'id'},
                {name: 'name'},
                {name: 'descn'}
            ])
        });
        ds.load();

        var grid = new Ext.grid.GridPanel({
            el: 'grid',
            ds: ds,
            cm: cm,
            layout:'fit'
        });

        var win = new Ext.Window({
            el:'window-win',
            layout:'fit',
            width:500,
            height:300,
            closeAction:'hide',

            items: [grid],

            buttons: [{
                text:'按钮'
            }]
        });
    win.show();
```

注意 items参数的值是一个数组，我们可以单独使用一个表格，也可以同时添加多个组件。

该示例在07.window\05-01.html中。

7.5.2 在窗口中加入表单

表单也可以直接放到窗口中。我们准备一个简单的表单来演示效果，实现过程如代码清单7-3所示。

代码清单7-3 设置表单

```
var form = new Ext.form.FormPanel({
    labelAlign: 'right',
    labelWidth: 50,
    frame: true,
    defaultType: 'textfield',
    items: [{
        fieldLabel: '文本',
        name: 'text'
```

```
        },{
            fieldLabel: '日期',
            name: 'data',
            xtype: 'datefield'
        }]
    });
```

其他配置基本不变，只是去掉了title参数，因为窗口自身提供了标题部分，在窗口中只显示一个组件的情况下，不需要重复设置标题。我们把width属性也去掉了，因为之后会把表单交给窗口进行布局管理，窗口中使用了layout:'fit'布局方式，最终表单会根据窗口的大小进行伸缩，不需要额外设置宽度了。

窗口的配置与上面相似，只是将items:[grid]改成items:[form]，这样就实现了在窗口中加入表单的效果，如图7-23所示。

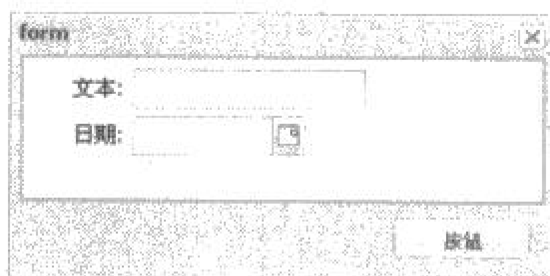


图7-23　窗口中的表单

该示例在07.window\05-02.html中。

7.5.3　复杂布局

图7-24显示了在窗口中显示多个组件的情况。

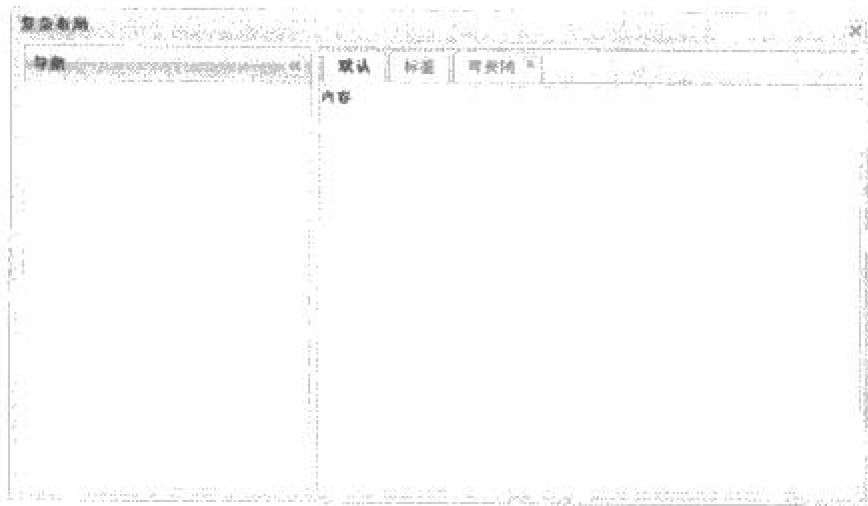


图7-24　复杂布局

在图7-24中，我们将窗口分隔成两个部分，左侧放置导航用的工具条，并且为工具条设置了折叠功能，折叠后的效果如图7-25所示。右侧显示的是一个包含3个面板的Ext.TabPanel，它的每个面板都有独立的标题和内容，第三个面板还可以由用户决定是否关闭。

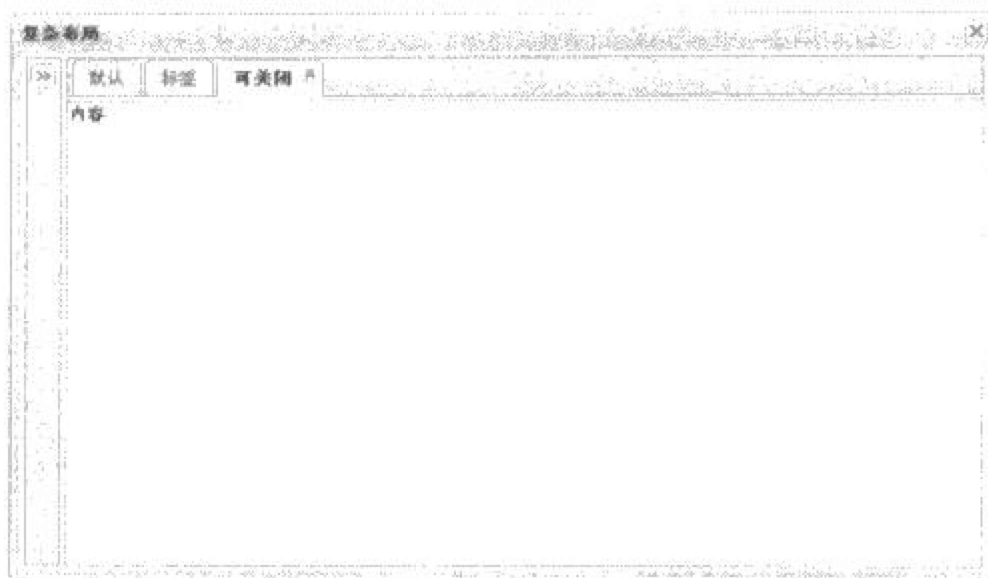


图7-25 面板折叠效果

该示例的实现过程如代码清单7-4所示。

代码清单7-4 窗口中的TabPanel

```
var tabs = new Ext.TabPanel({
    region: 'center',
    margins:'3 3 3 0',
    activeTab: 0,
    defaults:{autoScroll:true},

    items:[{
        title: '默认',
        html: '内容'
    },{
        title: '标签',
        html: '内容'
    },{
        title: '可关闭',
        html: '内容',
        closable:true
    }]
});

var nav = new Ext.Panel({
    title: '导航',
    region: 'west',
    split: true,
    width: 200,
    collapsible: true,
    margins:'3 0 3 3',
    cmargins:'3 3 3 3'
});

var win = new Ext.Window({
    title: '复杂布局',
```

```
        closable:true,
        width:600,
        height:350,
        border:false,
        layout: 'border',

        items: [nav, tabs]
    });

    win.show();
```

示例中为窗口指定了layout:'border'布局方式。我们将创建好的nav和tabs对象放到items参数中，窗口就会在显示时使用border边界布局方式控制nav和tabs的位置和大小。当窗口改变大小时，内部组件也会自动调整自身大小来适应窗口的变化。

有关布局的更多内容，请参考第8章。

7.6 小结

本章先介绍了如何使用Ext.MessageBox实现多种对话框，同时介绍了自定义对话框的方法：让对话框支持多行输入、自定义对话框的按钮、在对话框中显示进度条并为对话框提供动画效果等。

然后，讲解了Ext.Window的常用属性，包括创建简单窗口、控制窗口的最大化和最小化、配置窗口的关闭类型、控制窗口的移动范围、设置窗口中的按钮等。

最后讲解了窗口分组的实现方式，以及如何操作同一组中的多个窗口。此外，还讲解了窗口的内部布局功能，介绍了如何在窗口中显示表格和表单面板，以及如何在窗口中使用复杂布局方式。

第8章

布　　局

8

本章内容

- 布局的用途
- 最简单的布局FitLayout
- 常用的边框布局BorderLayout
- 制作伸缩菜单的布局Accordion
- 实现操作向导的布局CardLayout
- 控制位置和大小的布局AnchorLayout和AbsoluteLayout
- 表单专用的布局FormLayout
- 分列式布局ColumnLayout
- 表格状布局TableLayout
- 与布局相关的其他知识

8.1　布局的用途

所谓布局，简单来说就是决定把什么东西放到什么位置上。对管理软件来说，一般是首部放标题，左边放菜单栏，空余的右下方用来显示具体的内容，如图8-1所示。

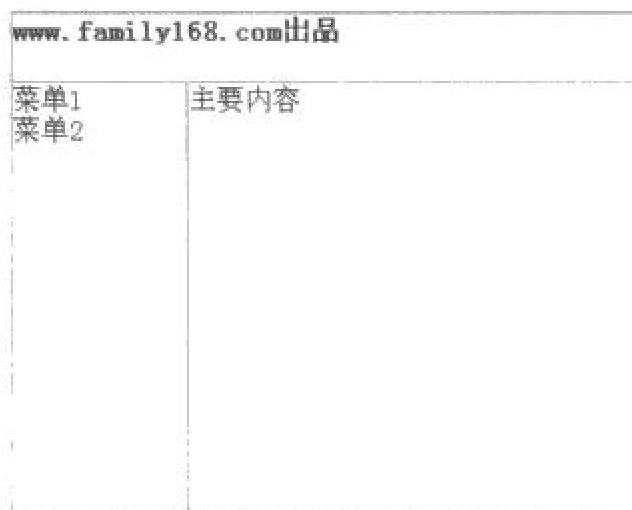


图8-1　简单布局示例

图8-1中的布局形式比较传统。在EXT中制作这种传统形式的布局比较简单，使用下面的配置内容就能实现，如代码清单8-1所示。

代码清单8-1 在EXT中实现传统布局

```
var viewport = new Ext.Viewport({
    layout: 'border',
    items: [{
        region: 'north',
        height: 40,
        html: '<h1>www.family168.com出品</h1>'
    },{
        region: 'west',
        width: 100,
        html: '<p>菜单1</p><p>菜单2</p>'
    },{
        region: 'center',
        html: '主要内容'
    }]
});
```

在上面的代码中，我们用Ext.Viewport类对整个页面进行了整体布局。这个类的具体用法稍后会详细解释，我们现在主要看看它里面是如何布局的。

Ext.Viewport里主要有两个配置参数：layout和items。其中layout:'border'表示我们使用了BorderLayout的布局方式，这种布局方式称为边界布局，它将页面分隔成东、西、南、北、中5个部分，我们使用region参数为它里面的组件指定了具体的放置位置。它里面的组件就定义在items参数里。items中定义了3个部分，分别是放在上方的region:'north'，左边的region:'west'和位于中间的region:'center'。从HTML部分的内容也可以看出页面上显示的部分究竟对应着代码中的哪些部分。

上面的代码实现了一个布局好的页面，完整的代码可以在01.html中找到。虽然它现在还略显粗糙，但也可以自动检测浏览器的大小变化和自动调整布局中每个部分的大小。你可以打开示例页面，用鼠标调整浏览器的大小，这时就会看到页面中的布局也会随着浏览器的大小实时变化。

在EXT中，所有的布局都是从Ext.Container开始的，Ext.Container的父类是Ext.BoxComponent。Ext.BoxComponent是一个盒子组件，可以定义宽度、高度和位置等属性。作为子类的Ext.Container也继承了父类的这些功能，更重要的是，Ext.Container可以使用layout和items属性为内部的子组件进行布局。

图8-2是Ext.Container以及它的子类的继承图。

根据继承原理，只要是Ext.Container的子类都可以使用layout对内部items进行布局。

事实上，我们经常用来设置布局的子类只有几个，如用Ext.Viewport进行页面的整体布局，使用Ext.Panel和Ext.Window进行各种嵌套布局，使用Ext.form.FieldSet和Ext.form.FormPanel为表单进行布局。其余的子类都使用默认的渲染形式，很少进行内部布局。

与Ext.Container相似，所有的布局类也有一个共同的超类Ext.layout.ContainerLayout。凡是继承自该超类的子类都可以对Ext.Container和它的子类进行布局定义，这两棵继承树结合在一起便构成了EXT中完整的布局系统，如图8-3所示。

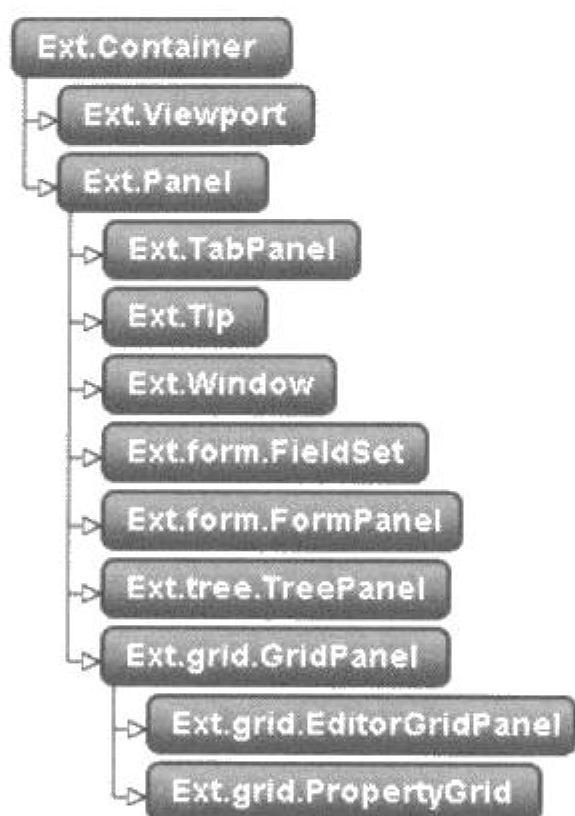


图8-2 Container及其子类继承图

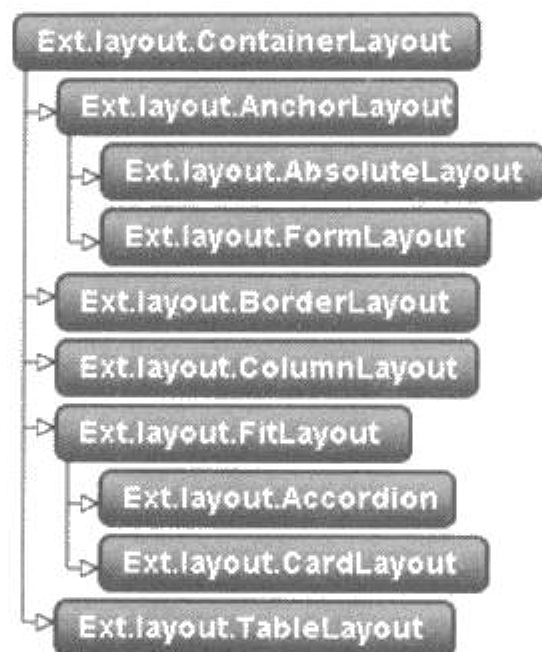


图8-3 布局类型继承图

上述示例中用到的边框布局Ext.layout.BorderLayout处于整个树形继承图的中间部位，其他常用到的布局还有用于表单布局的FormLayout、实现分列布局的ColumnLayout和自动填充整个布局空间的自适应FitLayout。除此之外，还有绝对位置布局AbsoluteLayout、折叠布局Accordion、锚点布局AnchorLayout、卡片布局CardLayout、容器布局ContainerLayout和表格布局TableLayout。

8.2 最简单的布局——FitLayout

有一个很简单的需求：客户需要页面里显示一个表格，让它可以自动适应页面大小的变化，页面变大的时候表格也会变大，页面变小的时候表格也会随之变小。

要想实现这样的功能，你是不是又会想到先去监听页面触发的事件，然后用得到的页面大小去修改表格呢？试试EXT提供的FitLayout吧，看看它是如何实现这个功能的，如下面的代码所示。

```
var viewport = new Ext.Viewport({
    layout: 'fit',
    items: [grid]
});
```

我们把Viewport中的layout修改为'fit'，再把预先创建好的表格放到items中，这样就能得到如图8-4的效果（见02-01.html）。

就这样，我们利用表格实现了预期的效果，表格填充了整个页面，并且它自身的大小会根据页面的大小而变化。Ext.layout.FitLayout虽然简单，但还是有几点需要注意。首先，使用了layout:'fit'组件的items只能放一个子组件。即使放了多个子组件，也只有第一个子组件会起作用。例如，我们将02-01.html修改为如下（见02-02.html）所示。

```
    var viewport = new Ext.Viewport({
        layout: 'fit',
         items: [{
            title: 'panel',
            html: 'panel'
},grid]
    });
```

如果像上面代码一样，在原来的表格前面添加一个panel，最后显示的结果就会如图8-5所示。

图8-4　FitLayout布局

图8-5　在FitLayout中设置多个panel

因为在表格的前面添加了panel，所以表格失效了，最后得到的页面中就只能看到排在最前面的panel。

还有一个问题就是在items中的表格里千万不要使用autoHeight:true参数，这个参数会使FitLayout失效。它会重新计算表格的高度，最后得到的表格也就无法填充整个页面了。使用了autoHeight:true的示例可以参考02-03.html和02-04.html，02-03.html的效果如图8-6所示。

02-03.html中表格的高度超过了页面高度，不但没有显示滚动条，连下面的分页工具条也被挤出了页面。

02-04.html的效果如图8-7所示。

图8-6　使用了autoHeight:true的示例（1）

图8-7　使用了autoHeight:true的示例（2）

在02-04.html中，表格的高度小于页面高度，整个表格收缩了起来，没有填满整个页面。因为参数autoHeight是定义在Ext.BoxComponent中的，继承了它的子类都可能会遇到图8-7中的问题，所以当布局出现问题时，可以考虑检查一下是不是由于这个参数造成的。

因为之前的示例都是使用Ext.Viewport进行整个页面范围的布局控制，所以最后我们提供一个在Ext.Window中用于局部布局的示例供大家参考，如下面的代码所示。

```
var win = new Ext.Window({
    width: 400,
    height: 300,
    layout: 'fit',
    items: [grid]
});
```

从上面的代码可以看出，layout和items的部分基本上是相同的，只是将原来的类名由Ext.Viewport换成了Ext.Window，并且再加上了Ext.Window需要的width和heigth参数。这样就成功地把原来放在Ext.Viewport中的表格移植到了Ext.Window窗口中，以后表格将会适应Ext.Window的大小变化，而不是根据整个页面的大小调整自己的大小。

该示例在02-05.html中，运行结果如图8-8所示。

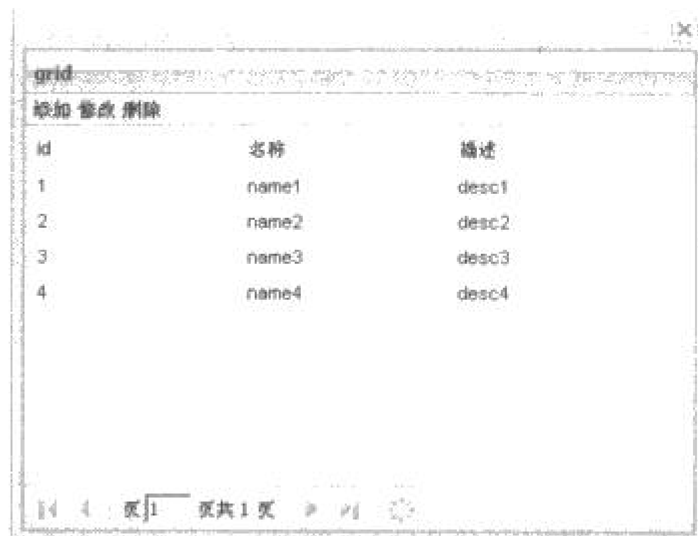


图8-8 在Ext.Window中使用FitLayout

8.3 常用的边框布局——BorderLayout

因为FitLayout布局每次只能使用一个子组件，根本无法胜任更复杂的布局需求，所以现实中我们使用得最多的是Ext.layout.BorderLayout布局。它将整个布局区域分成了东、西、南、北、中5个部分。除了中间区域以外，其他的区域又都是可以省略的，因此我们可以利用它制作出更复杂、更灵活的布局。

在本章的第一个示例中，我们就已经见过layout:'border'以及region:'center'制作的效果了。这里再实现一个完整的示例，如代码清单8-2（见03-00.html）所示。

代码清单8-2 用BorderLayout实现一个完整的布局

```
var viewport = new Ext.Viewport({
    layout: 'border',
    items: [
        {region:'north',html:'north'},
        {region:'south',html:'south'},
        {region:'east',html:'east'},
        {region:'west',html:'west'},
        {region:'center',html:'center'}
    ]
});
```

这段代码的显示效果如图8-9所示。

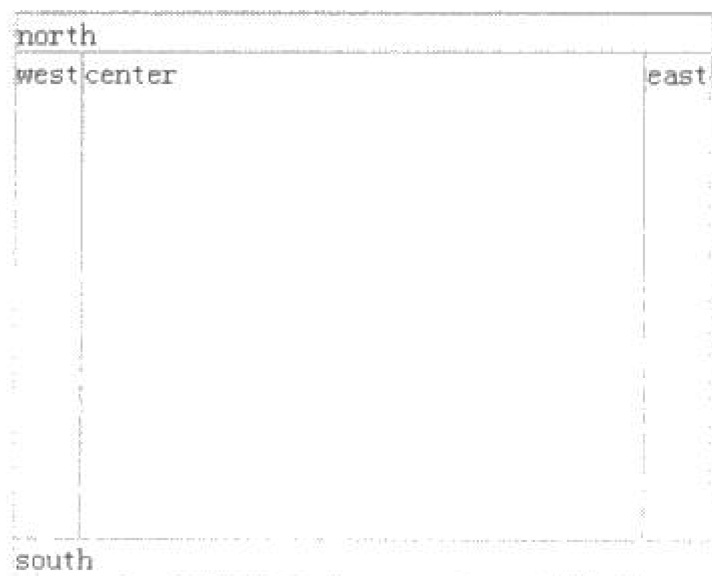


图8-9　使用了BorderLayout的布局

对BorderLayout来说，5个部分是预先就设置好的，根据上北、下南、左西、右东的方式进行布局。其中north和south分别位于页面的最上方和最下方，可以分别用来做系统的标题和状态栏；west和east分别位于页面的左边和右边，是为菜单工具条准备的；center的大小是在其他4个部分设置好以后自动计算出来的，它也是唯一不能省略的部分。

再重复一遍，center是绝对不能省略的。如果items中缺少了region:'center'就会报错，会造成程序中断，页面上就什么也看不到了。

了解了BorderLayout的基本用法以后，我们可以开始进一步挖掘它更深层次的功能。你会了解到BorderLayout并不是只有空空的5个分区，利用它的附加功能我们可以打造出更完美的布局。

8.3.1　设置子区域的大小

这里所谓的子区域就是设置north、south、west、east 4个区域，不包括中间的center，因为中间这部分的大小是通过其他部分计算得来的。

从图8-9中也可以看出，north和south部分只能设置高度（height），west和east部分只能设置宽度（width）。虽然在使用HTML时它会自动计算适合本身的尺寸大小并进行显示，但在这里我们还是说说如何通过指定它们的宽度和高度来控制显示效果。

如果为图8-9中的示例指定宽度和高度，结果会如图8-10所示。

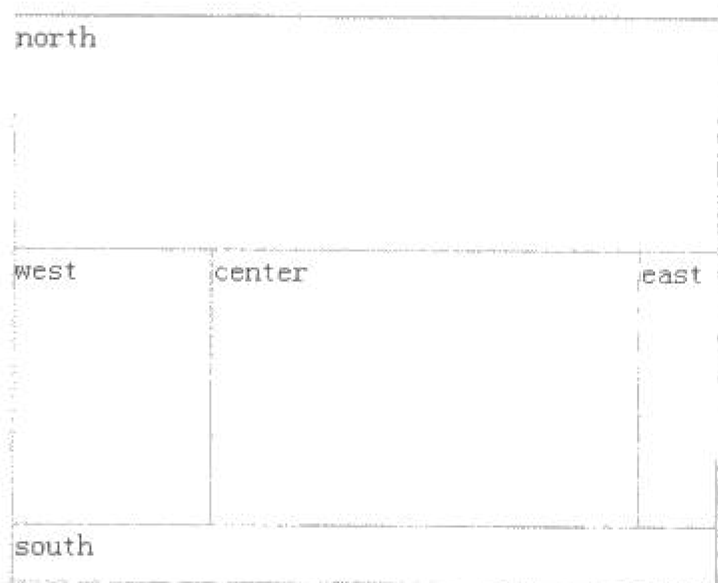


图8-10　设置子区域的宽度和高度

从图8-10中可以看出，north和west区域变大了，center区域却变小了。实际上我们也稍稍增大了east和west部分，只是改变的幅度太小，不容易发现罢了。现在我们来看看到底如何控制各个区域的大小，如代码清单8-3（见03-01-01.html）所示。

代码清单8-3　指定各子区域的大小

```
var viewport = new Ext.Viewport({
    layout: 'border',
    items: [
        {region:'north',html:'north',height:120},
```

```
            {region:'south',html:'south',height:30},
            {region:'east',html:'east',width:40},
            {region:'west',html:'west',width:100},
            {region:'center',html:'center'}
        ]
    });
```

从上述代码中可以看出，我们只添加了width和height参数，就可以指定每个子区域的大小了。但是，正如前面所讲过的，north和south两个区域只能指定高度值，宽度值由BorderLayout自动计算；east和west只能指定宽度值，高度值由BorderLayout自动计算；center区域的大小是由其他4个部分决定的，所以不能为它指定宽度值或高度值。

另外，我们再来看一下autoHeight:true在布局方面的缺点。原本在BorderLayout中的子组件都是自动延展的，north和south部分即使设置了宽度也不会起作用。同理，即使为west或east设置了高度也不会影响布局的结果。但是autoHeight:true的结果却是破坏性的，如果设置了一个或多个autoHeight:true，就足以毁灭整个布局的成果。例如，我们给所有的子组件都加上autoHeight:true，代码如下所示：

```
    var viewport = new Ext.Viewport({
        layout: 'border',
        items: [
            {region:'north',html:'north',height:120,autoHeight:true},
            {region:'south',html:'south',height:30,autoHeight:true},
            {region:'east',html:'east',width:40,autoHeight:true},
            {region:'west',html:'west',width:100,autoHeight:true},
            {region:'center',html:'center',autoHeight:true}
        ]
    });
```

结果如图8-11所示。

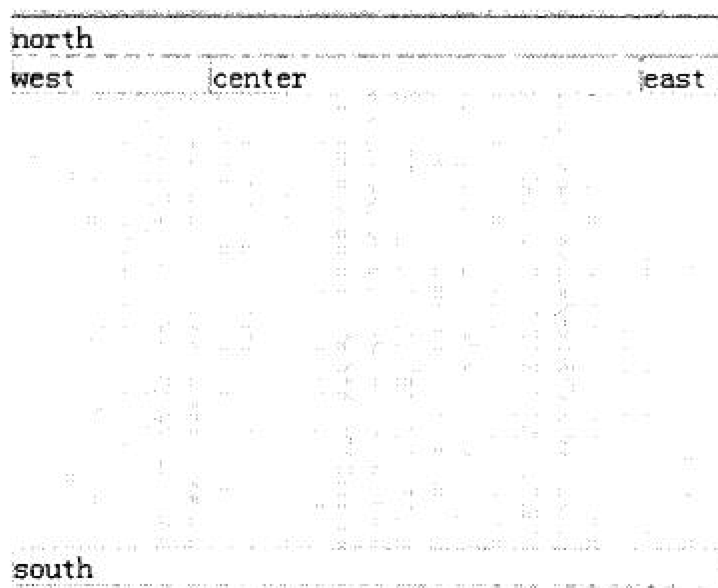


图8-11 BorderLayout使用autoHeight:true

该示例在03-01-02.html中。

8.3.2　使用 split 并限制它的范围

使用了split就允许用户自行拖放以改变某一个区域的大小。如图8-12所示，允许用户修改west区域的宽度。

从图8-12中可以看出，west和center区域相交的边界分割线变宽了。现在我们可以把鼠标移动到这条边界上，通过拖放改变west区域的宽度。当然，center区域的宽度也会随之改变。拖放后的效果如图8-13所示。

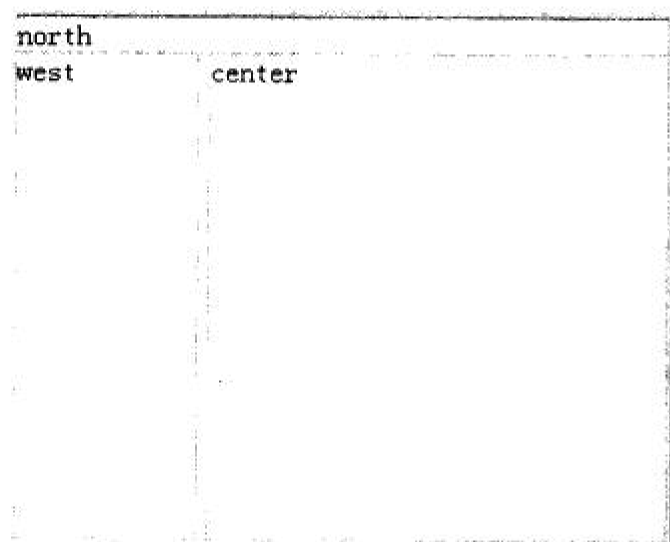


图8-12　使用split:true

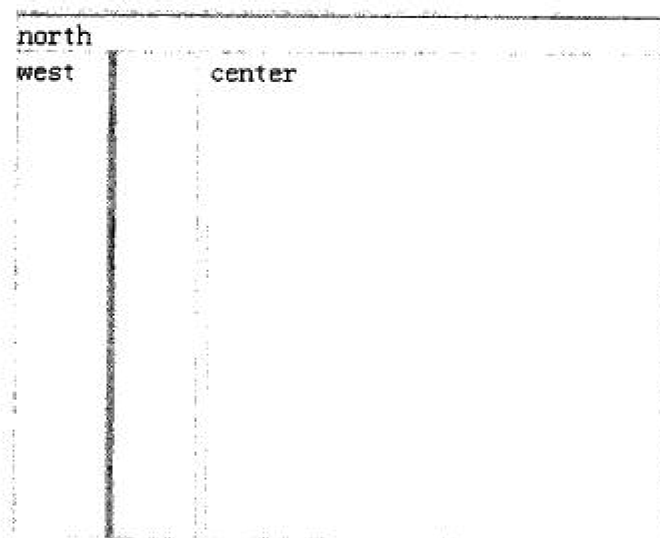


图8-13　减小west区域的宽度

接下来看看下面的代码（见03-02-01.html），只是为west区域添加了split:true参数。

```
var viewport = new Ext.Viewport({
    layout: 'border',
    items: [
        {region:'north',html:'north'},
        {region:'west',html:'west',width:100,split:true},
        {region:'center',html:'center'}
    ]
});
```

需要注意的是，即使加上了split:true参数，north和south区域也只能上、下拖放，west和east区域只能左、右拖放。center区域则不会变化，即使给center加上split:true也不会出现任何拖放边界。图8-14就是为north部分加上了split:true参数后的效果，示例在03-02-02.html中。

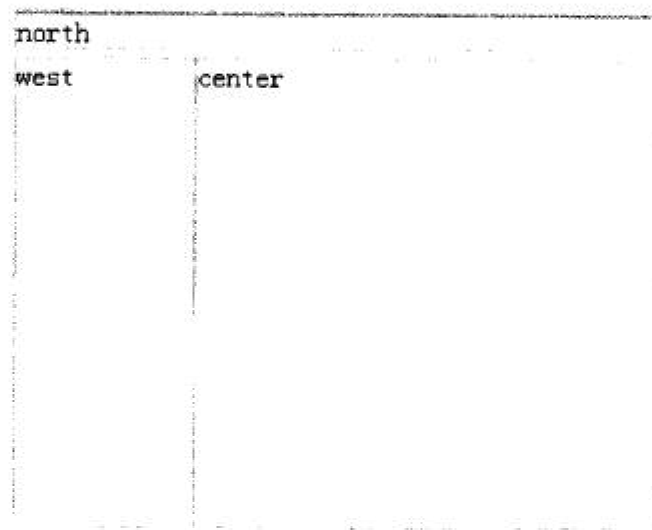


图8-14　为north部分设置split:true

再次提醒大家：有时用户输入的数据是不可信的，无法预知用户的操作会导致哪些问题，所以必须限制用户的操作。我们为用户提供了拖放功能，就要限制可以拖放的范围，否则用户在拖放过程中很可能会导致布局变得很乱。在出现问题时，用户

也许还会认为是程序代码在某些方面不够严谨造成的。

我们首先要限制左侧菜单区域（west）的大小，不允许它太窄，以至于无法完整显示菜单的内容；也不允许它太宽，以至于整个布局变形。这时，我们就需要使用参数minSize和maxSize。将这两个参数与split:true结合使用，就可以限制用户拖放的范围了，如下面的代码（03-02-03.html）所示。

```
var viewport = new Ext.Viewport({
    layout: 'border',
    items: [
        {region:'north',html:'north'},
        {region:'west',html:'west',width:100,
        split:true,minSize:80,maxSize:120},
        {region:'center',html:'center'}
    ]
});
```

上面的代码指定了west的初始宽度为100，然后让west的宽度只可以在80和120之间变化，不能超出这个范围，以此控制用户的操作。

在north里也可以是使用minSize和maxSize，只不过这里控制的是最小高度和最大高度，参数的配置方法相似，我们这里就不详细介绍了。如果有问题，可以参考03-02-04.html中的示例。

8.3.3 子区域的展开和折叠

该功能可以让一个区域展开和折叠（相当于隐藏），效果如图8-15所示。

单击west区域的折叠按钮，结果如图8-16所示。

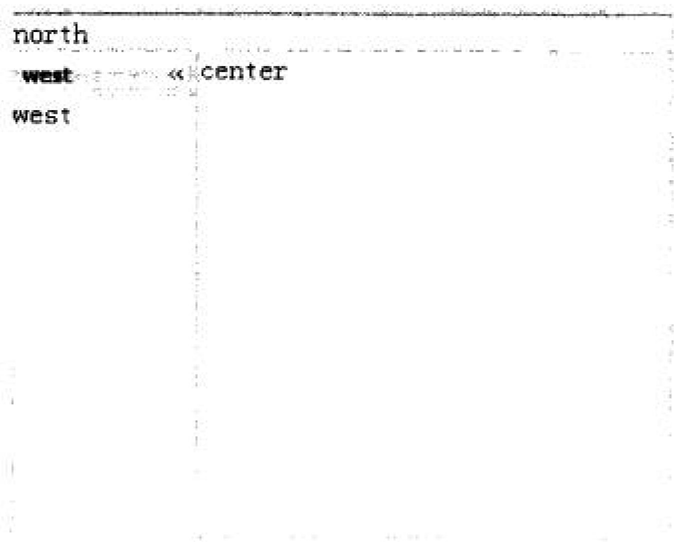


图8-15 展开后的west区域

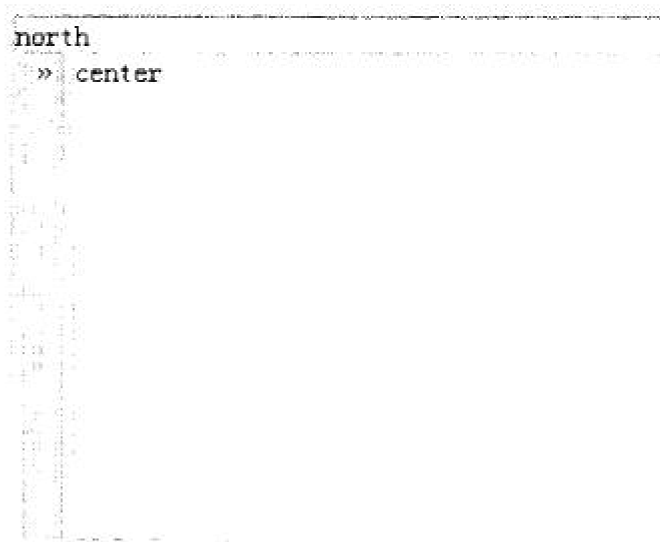


图8-16 折叠后的west区域

要实现这个效果，只需要配置几个参数即可，如下面的代码（见03-03-01.html）所示。

```
var viewport = new Ext.Viewport({
    layout: 'border',
    items: [
        {region:'north',html:'north'},
        {region:'west',html:'west',title:'west',
```

```
width:100,collapsible:true},
        {region:'center',html:'center'}

    ]
});
```

主要的配置参数是collapsible:true，这个参数激活了west区域的折叠功能，而前面的title参数也是必须设置的，因为折叠按钮是出现在标题部分的。如果没有为west区域设置标题，折叠按钮也就无法显示了，所以一定要加上title参数。

为north区域设置折叠功能的配置方式与west完全相同，只是north会向上折叠。你甚至还可以为center区域设置折叠功能，不过center部分即使折叠了，也不会影响其他部分的大小，如图8-17（见03-03-02.html）所示。

如果将每个区域都折叠起来，效果如图8-18所示。

要实现这种效果，代码中的参数设置与前面所讲的完全相同。要记住，当设置collapsible:true时，必须保证title参数也设置好，这样就能正常显示所有折叠按钮了。

到这里，我们已经可以完全实现BorderLayout中每个部分的折叠和展开功能。不过，如果折叠后的区域什么都不显示，那么就不便查看该区域里的内容了。如果可以在折叠的列上添加提示信息，那么查看该区域里的内容就比较方便了。

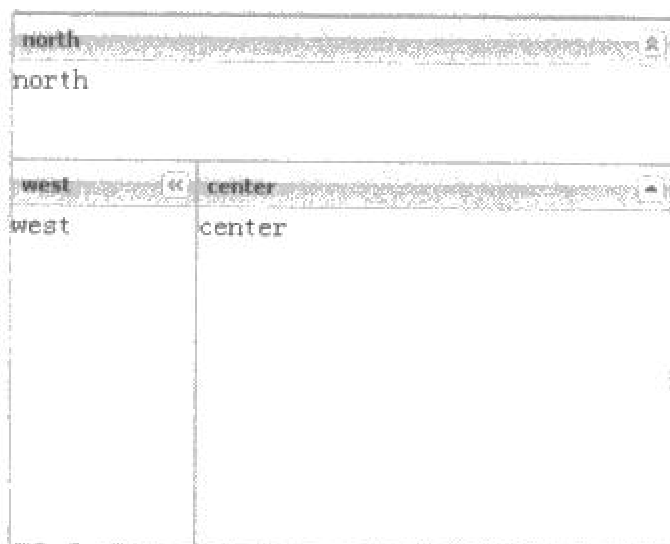


图8-17　将多个区域设置为可折叠的

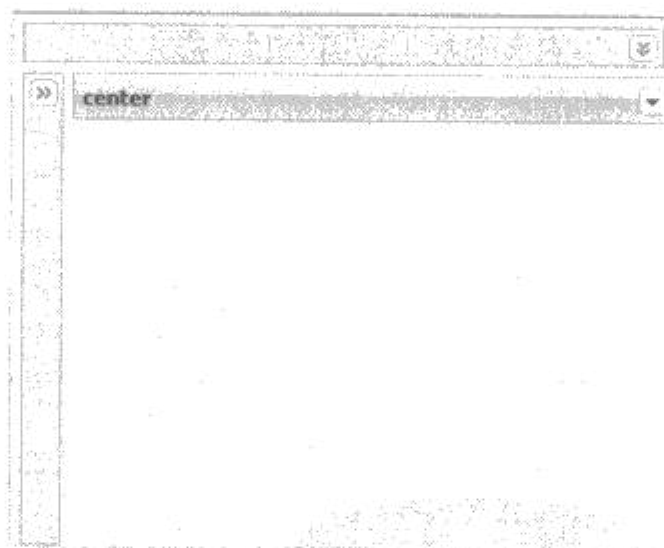


图8-18　所有区域被折叠后的效果

下面我们来看看折叠后显示提示信息的方法。因为HTML中是不支持文字竖排的，所以west和east这样的细长形状的区域就无法用设置文本的方式设置提示信息。EXT中也没有为折叠后的提示信息设置默认的实现方式。因此，我们这里的方法就是修改各区域折叠后的CSS样式，通过设置背景图片的方式来实现添加提示信息的目的。

为此，我们先为各区域设置对应的CSS样式，如代码清单8-4所示。

代码清单8-4　设置各区域的提示信息的CSS样式

```
.x-layout-collapsed-north {
    background-image: url(user_male.png);
    background-repeat: no-repeat;
```

```
        background-position: center;
    }
    .x-layout-collapsed-south {
        background-image: url(user_female.png);
        background-repeat: no-repeat;
        background-position: center;
    }
    .x-layout-collapsed-west {
        background-image: url(user_male.png);
        background-repeat: no-repeat;
        background-position: center;
    }
    .x-layout-collapsed-east {
        background-image: url(user_female.png);
        background-repeat: no-repeat;
        background-position: center;
    }
```

这里的CSS样式类名是默认的，与各区域的名称一一对应。样式中我们设置了3个属性：background-image表示背景图片的路径，background-repeat表示背景图片只会显示一次，background-position表示背景图片会居中显示。因此，如果想显示文本内容，可以制作文字图片。

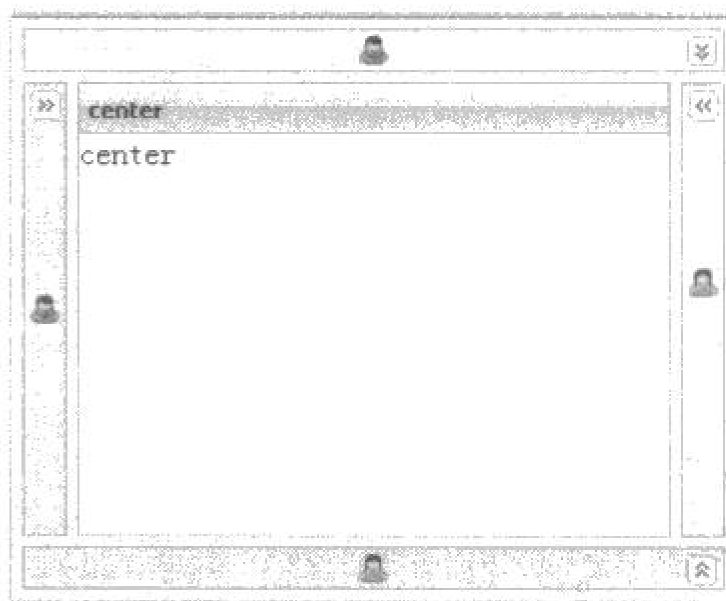


图8-19 添加了提示信息图片后的折叠效果

将这些CSS样式添加到页面中，不需要对JavaScript代码部分进行修改，就可以得到图8-19的折叠效果。

当然，图8-19中显示的提示信息图片过于简单。实际中应用中，大家还要考虑图片的具体高度和宽度，因为图片不会根据布局调整自己的大小。示例代码如下所示：

```
    var viewport = new Ext.Viewport({
            layout: 'border',
            items: [
                {region:'north',html:'north',title:'north',height:80,collapsible:true},
                {region:'south',html:'south',title:'south',height:80,collapsible:true},
                {region:'west',html:'west',title:'west',width:100,collapsible:true},
                {region:'east',html:'east',title:'east',width:100,collapsible:true},
                {region:'center',html:'center',title:'center'}
            ]
        });
```

至此，BorderLayout的主要功能都已经讲完了。只要掌握了上面的内容，大家就可以实现丰富而灵活的布局样式了。下面我们将讲述另外几种布局方式，它们都可以与BorderLayout结合使用，从而实现功能更为复杂的布局样式。

8.4　制作伸缩菜单的布局——Accordion

用过QQ的朋友应该会对它里面的伸缩菜单有深刻的印象。展开折叠分组，再配上动画效果，我相信很多人都想实现这样的画面和效果。

Accordion是EXT中默认布局的一部分，如果想用它，直接将区域加上`layout:'accordion'`即可，其他部分基本无需改动。

我们先来看看图8-20中的示例。

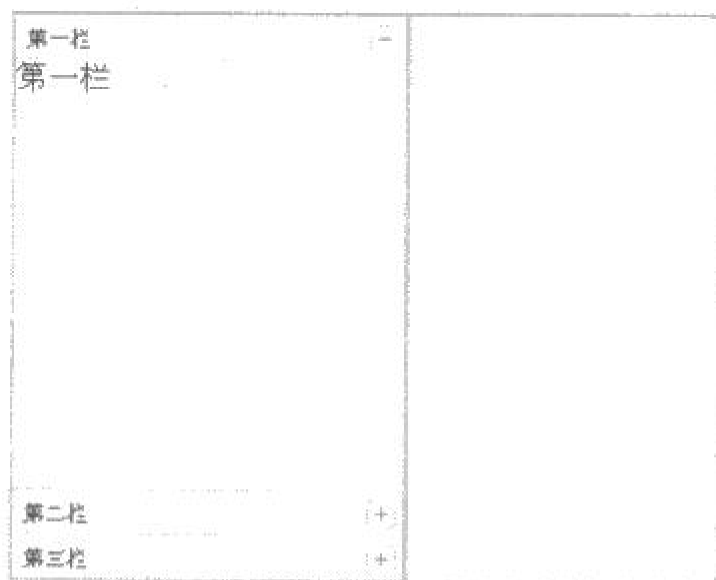


图8-20　使用了Accordion的示例

我们利用`ViewPort`制作出只有west和center两个区域的BorderLayout，在west部分放上Accordion，实现方法如代码清单8-5所示。

代码清单8-5　Accordion示例

```
var viewport = new Ext.Viewport({
    layout:'border',
    items:[{
        region: 'west',
        width: 200,
        layout: 'accordion',
        layoutConfig: {
            titleCollapse: true,
            animate: true,
            activeOnTop: false
        },
        items: [{
            title: '第一栏',
            html: '第一栏'
        },{
            title: '第二栏',
            html: '第二栏'
        },{
            title: '第三栏',
```

```
                html: '第三栏'
            }]
        },{
            region: 'center',
            split: true,
            border: true
        }]
    });
```

设置了`layout:'accordion'`后，再使用`items`添加3个元素，记得每个子元素里都要加上`title`参数，Accordion没有提供默认的标题，不设置标题是一定会出错的。

与布局有关的配置参数都写到`layoutConfig`里了，这也是在`Ext.Conainer`中定义好的。与布局有关的所有配置项都写在`layoutConfig`中了，随后在进行布局时会自动赋给对应的`layout`实例，并产生作用。这些配置项如下所示。

- `titleCollapse`：默认为`true`，单击标题就可以折叠子面板；如果设置为false，就只能通过单击标题右边的图标折叠子面板。
- `animate`：展开和折叠时是否使用动画效果。
- `activeOnTop`：默认值是`false`，执行展开和折叠操作后，子面板的顺序不会改变。如果设置为`true`，就会随着展开和折叠的顺序而改变，展开的子面板总是放在最上面。

该示例在04.html中。

8.5 实现操作向导的布局——CardLayout

CardLayout可以看作是一叠卡片，从上面看起来就像是一张卡片，可以把中间的卡片抽出来，放到最上面，但每次只能看到一张卡片。

这种布局特性用来制作操作向导最为合适，图8-21就是一个有3个步骤的操作向导。

单击“下一步”，结果如图8-22所示。

再单击“下一步”，操作向导就结束了，如图8-23所示。

就像我们所见到的那样，虽然给CardLayout配置了几个子面板，但它每次只显示其中一个，看上去有点儿像幻灯片。布局倒是很简单，只要改成`layout:'card'`就可以了，如代码清单8-6所示。



图8-21 CardLayout第一步



图8-22 CardLayout第二步

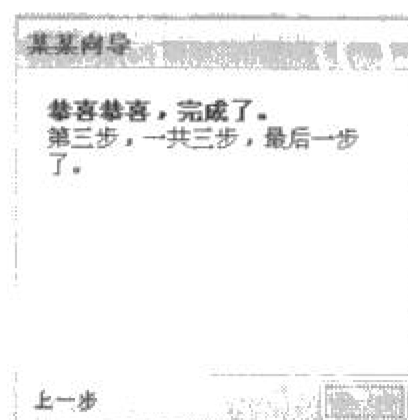


图8-23 CardLayout第三步

代码清单8-6　CardLayout

```
var viewport = new Ext.Viewport({
    layout:'border',
    items:[{
        region: 'west',
        id: 'wizard',
        width: 200,
        title: '某某向导',
        layout:'card',
        activeItem: 0,
        bodyStyle: 'padding:15px',
        defaults: {
            border:false
        },
        bbar: [{
            id: 'move-prev',
            text: '上一步',
            handler: navHandler.createDelegate(this, [-1]),
            disabled: true
        }, '->',{
            id: 'move-next',
            text: '下一步',
            handler: navHandler.createDelegate(this, [1])
        }],
        items: [{
            id: 'card-0',
            html: '<h1>哈哈！<br />欢迎用咱的向导。</h1><p>第一步，一共三步</p>'
        },{
            id: 'card-1',
            html: '<p>第二步，一共三步</p>'
        },{
            id: 'card-2',
            html: '<h1>恭喜恭喜，完成了。</h1><p>第三步，一共三步，最后一步了。</p>'
        }]
    },{
        region: 'center',
        split: true,
        border: true
    }]
});
```

当前显示的子面板是用activeItem来控制的，在activeItem初始值为0时，是告诉CardLayout，要显示items中索引为0的子面板，也就是id:'card-0'的部分。

bbar里设置的是两个按钮，控制上下翻页的代码（handler）稍微有些复杂。首先判断需要显示哪一个子面板，得到索引值后就调用CardLayout的setActiveItem()函数把这个子面板显示出来。剩下的问题就是如何根据当前页面的索引位置改变前进和后退按钮的状态，代码如下所示：

```
var navHandler = function(direction){

        var wizard = Ext.getCmp('wizard').layout;
        var prev = Ext.getCmp('move-prev');
        var next = Ext.getCmp('move-next');
```

```
        var activeId = wizard.activeItem.id;

        if (activeId == 'card-0') {
            if (direction == 1) {
                wizard.setActiveItem(1);
                prev.setDisabled(false);
            }
        } else if (activeId == 'card-1') {
            if (direction == -1) {
                wizard.setActiveItem(0);
                prev.setDisabled(true);
            } else {
                wizard.setActiveItem(2);
                next.setDisabled(true);
            }
        } else if (activeId == 'card-2') {
            if (direction == -1) {
                wizard.setActiveItem(1);
                next.setDisabled(false);
            }
        }
    };
```

该示例在05.html中。

8.6 控制位置和大小的布局——AnchorLayout 和 AbsoluteLayout

AnchorLayout提供了一种灵活的布局方式，既可以为items中的每个组件指定与总体布局大小的差值，也可以设置一个比例使子组件可以根据整体自行计算本身的大小。它为我们提供了3种配置方式。

第一种方式是使用百分比进行配置，我们可以设置某一个子组件占整体长和宽的百分比。例如，这里我们设置panel 1的宽度占整体宽度的50%，高度也占整体高度的50%；设置panel 2的宽度占整体宽度的80%，显示效果如图8-24所示。

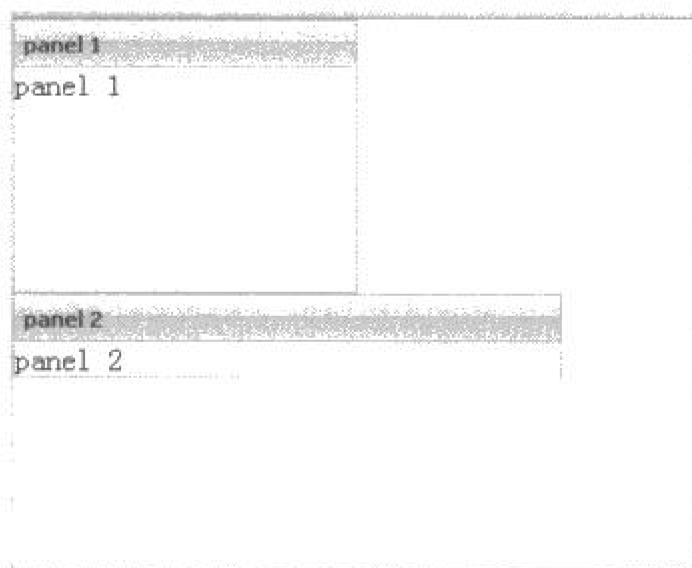


图8-24 在AnchorLayout中使用百分比的方式设置子组件的大小

图8-24的实现方法如代码清单8-7（见06-01.html）所示。

代码清单8-7　在AnchorLayout中使用百分比的方式设置子组件的大小

```
var viewport = new Ext.Viewport({
    layout:'anchor',
    items:[{
        title: 'panel 1',
        html: 'panel 1',
        anchor: '50% 50%'
    },{
        title: 'panel 2',
        html: 'panel 2',
        anchor: '80%'
    }]
});
```

从代码中可以看出，我们为panel 1设置了一个anchor参数，参数值是一个字符串，包含了两个用空格分隔的百分数。这两个百分数分别代表了所占的宽度和高度。Panel 2与之类似，只是anchor参数中只有一个百分数（80%）。这样配置的结果是，宽度设置为整体的80%，而高度自动设置为auto。

第二种方式是直接设置与右侧和底部的边距，效果如图8-25所示。

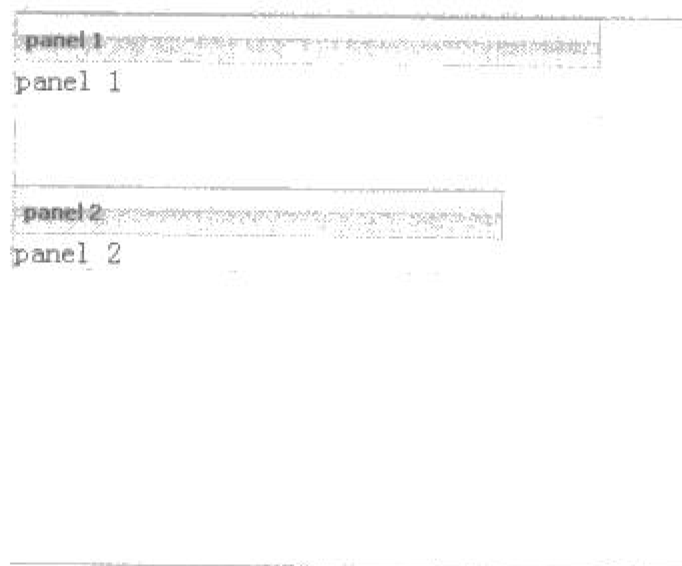


图8-25　设置与右侧和底端的边距

我们指定panel 1与右侧相距50像素，与底部相距200像素；panel 2与右侧相距100像素，如代码清单8-8（见06-02.html）所示。

代码清单8-8　在AnchorLayout中使用设置边距的方式设置子组件的大小

```
var viewport = new Ext.Viewport({
    layout:'anchor',
    items:[{
        title: 'panel 1',
        html: 'panel 1',
        anchor: '-50 -200'
```

```
        },{
            title: 'panel 2',
            html: 'panel 2',
            anchor: '-100'
        }]
    });
```

因为边距以像素为单位，所以anchor中不需要使用百分数。panel 1里依然有由空格分开的两个整数，表示与右侧和底部的相对距离，在anchor中使用负数表示子组件的实际大小要在整体的大小上减去对应的anchor值来得到。panel 2中只设置一个值，只会计算相对右侧的距离，高度自动赋予auto。

第三种方式称为side。使用它的前提是，为作为布局整体的父组件和布局内部的子组件都设置好width、height和anchorSize属性。AnchorLayout会记录布局整体与子组件在大小上的差值，为以后调整布局提供依据。

我们先来看看代码清单8-9（见06-03.html），如下所示。

代码清单8-9　在AnchorLayout中使用side方式设置子组件的大小

```
var viewport = new Ext.Viewport({
    layout:'anchor',
    anchorSize: {width:400,height:300},
    items:[{
        title: 'panel 1',
        html: 'panel 1',
        width: 200,
        height: 100,
        anchor: 'r b'
    },{
        title: 'panel 2',
        html: 'panel 2',
        width: 100,
        height: 200,
        anchor: 'r b'
    }]
});
```

我们为Viewport设置了anchorSize，这是一个包含宽度和高度信息的JSON对象，以此作为以后计算差值的基准。然后，分别在panel 1和panel 2中设置宽度和高度的信息。anchor的设置只有anchor:'r b'一种，这是固定写法，也可以写成anchor:'right buttom'，两种写法的效果是完全相同的。设置anchor:'r b'前的效果如图8-26所示，设置anchor:'r b'后的效果如图8-27所示。

设置了side之后的计算方式是这样的，AnchorLayout首先获得父组件的宽度、高度，以及每个子组件的宽度、高度，然后将子组件与父组件的宽度、高度之差分别保存起来。在父组件的大小改变之后，AnchorLayout会使用之前保存的差值，根据改变后父组件的大小，计算出子组件当前的宽度和高度。这种配置方法比较愚笨，实际使用时很少用到。

实际上，前两种配置方式已经很灵活了，我们还可以在anchor中同时使用百分比和边距的配置方式，如代码清单8-10所示。

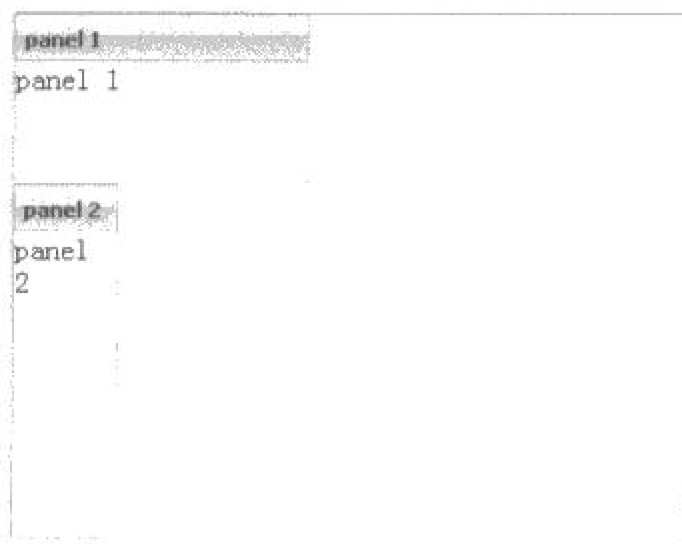


图8-26　在AnchorLayout中使用side方式设置子组件的大小（1）

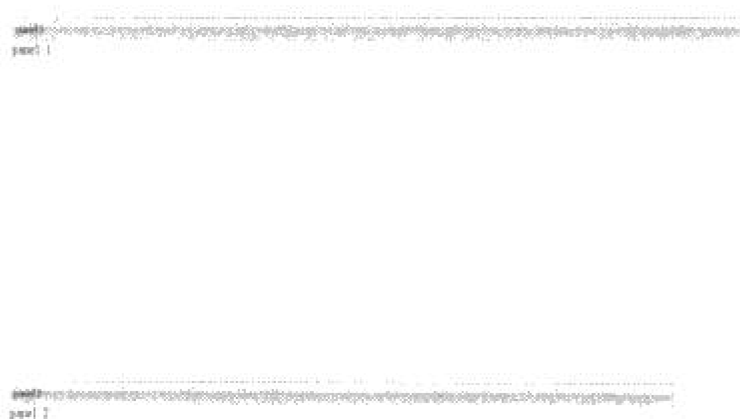


图8-27　在AnchorLayout中使用side方式设置子组件的大小（2）

代码清单8-10　AnchorLayout 组合配置形式

```
var viewport = new Ext.Viewport({
    layout:'anchor',
    items:[{
        title: 'panel 1',
        html: 'panel 1',
        anchor: '-100 40%'
    },{
        title: 'panel 2',
        html: 'panel 2',
        anchor: '-200 60%'
    }]
});
```

结果如图8-28所示。

这样，我们更容易对宽度不变、高度需要自由改变的布局进行精确的控制。

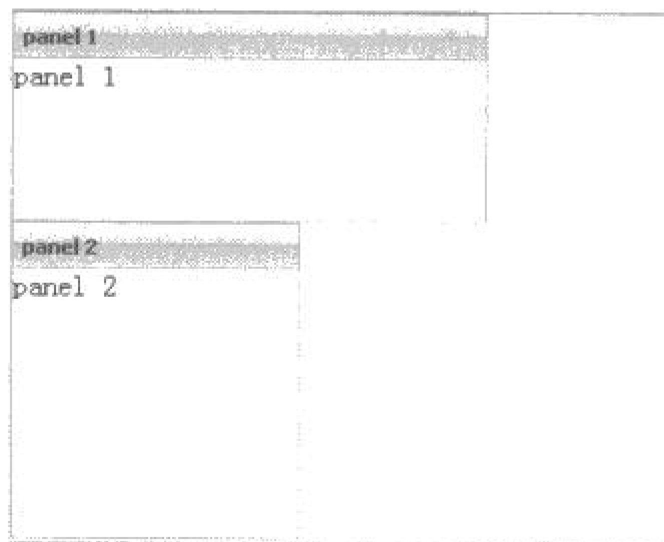


图8-28　同时使用百分比和边距两种配置方式

了解了AnchorLayout的应用后，我们就可以看看AbsoluteLayout的用法。AbsoluteLayout是AnchorLayout的一个子类，继承了AnchorLayout的所有特性，而且还有很多其他功能。

从上面的示例中我们可以看出，AnchorLayout布局下的子组件都是自上而下竖直排列的，我们不能控制每个组件的位置。AbsoluteLayout正是为解决这个问题而来的，我们可以通过设置x、y参数达到控制子组件位置的效果，如图8-29所示。

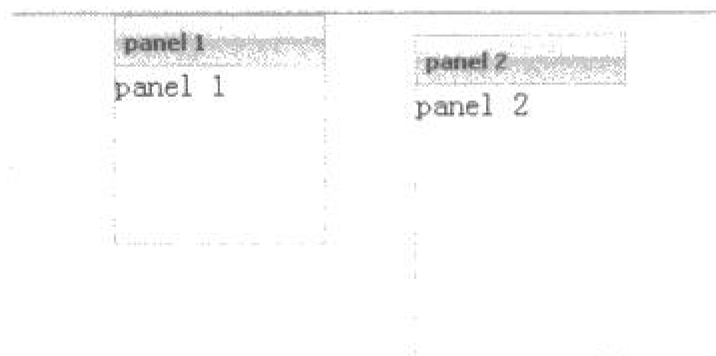


图8-29 使用AbsoluteLayout实现绝对定位

通过上面的讨论，我们已经可以通过AbsolutLayout为组件进行绝对定位，并使用AnchorLayout确定各个组件的相对大小。代码如下所示：

```
var viewport = new Ext.Viewport({
    layout:'absolute',
    items:[{
        title: 'panel 1',
        html: 'panel 1',
        x: 50,
        y: 0,
        anchor: '-200 40%'
    },{
        title: 'panel 2',
        html: 'panel 2',
        x: 200,
        y: 10,
        anchor: '-50 60%'
    }]
});
```

该示例在08.layout/06-05.html中。

8.7 表单专用的布局 FormLayout

FormLayout也是AnchorLayout的一个子类，可以在它里面使用anchor设置宽和高的比例。但是，它主要还是用于对表单进行布局。首先要明白的一点是，Ext.form.FormPanel使用它作

为默认的布局方式。也正因为使用了FormLayout布局，我们设置的fieldLabel、boxLabel才能显示出来，而FormLayout里也提供了一系列的参数来控制这些显示效果。

本章主要讲解与布局有关的知识，因此我们来看看与表单有关的配置参数，然后实现一个使用anchor控制多个Ext.form.Field的实例。

FormLayout提供的用于控制表单显示的参数如表8-1所示。

表8-1 FormLayout提供的用于控制表单显示的参数

参 数 名	描 述
hideLabels	是否隐藏控件的标签
itemCls	表单显示的样式
labelAlign	标签的对齐方式（左、中、右）
labelPad	标签空白的像素值
labelWidth	标签宽度

FormLayout为Ext.form.Field提供的配置参数如表8-2所示。

表8-2 Ext.form.Field对应的参数

参 数 名	描 述
clearCls	清除 DIV 渲染的 CSS 样式
fieldLabel	对应控件的标签内容
hideLabel	是否隐藏标签
itemCls	控件的 CSS 样式
labelSeparator	标签和控件之间的分隔，默认是“:”
labelStyle	标签的 CSS 样式

以上这些是FormLayout与Ext.form.BasicForm和Ext.form.Field对应的配置参数，详细内容请参考第4章。

下面我们来演示一下如何使用anchor对表单内的多个field进行布局，效果如图8-30所示。

实现的效果是，每个field的宽度占整体宽度的90%，而且宽度可以根据页面大小的变化自动调整。“备注”对应的textarea的高度也是可以自动延展的。这样使用anchor设置大小，不仅避免了考虑每个组件的宽度和高度（这样很麻烦），还可以实现自动调整大小，何乐而不为呢？实现过程如代码清单8-11所示（见07.html）。

图8-30 利用Formlayout对表单内的多个field进行布局

代码清单8-11 利用Formlayout对表单内的多个field进行布局

```
var viewport = new Ext.Viewport({
    layout:'fit',
    items:[{
        xtype: 'form',
        title: '信息',
        labelAlign: 'right',
        labelWidth: 50,
        frame: true,
        defaultType: 'textfield',
        items: [{
            fieldLabel: '名称',
            name: 'name',
            anchor: '90%'
        }, {
            fieldLabel: '生日',
            name: 'birthday',
            xtype: 'datefield',
            anchor: '90%'
        },{
            fieldLabel: '备注',
            name: 'desc',
            xtype: 'textarea',
            anchor: '90% -100'
        }]
    }]
});
```

之前的示例大多数是使用width和height手工控制field的大小，希望大家能尝试使用布局的方式对表单进行控制。

8.8 分列式的布局 ColumnLayout

在第4章中我们已经了解了使用ColumnLayout进行分列布局的相关知识。在此将进一步学习它，这次的重点不是表单的排列，而是对ColumnLayout的基本用法的研究。

先看一个简单的分列布局示例，如图8-31所示。



图8-31 使用了ColumnLayout的简单分列布局

实现过程如代码清单8-12（见08-01.html）所示。

代码清单8-12 使用ColumnLayout实现简单分列布局

```
var viewport = new Ext.Viewport({
    layout:'column',
    items: [{
        title: 'Column 1',
        columnWidth: .25
    },{
        title: 'Column 2',
        columnWidth: .4
    },{
        title: 'Column 3',
        columnWidth: .35
    }]
});
```

请注意items中每个子组件的columnWidth参数，它是0到1之间的一个小数，表示每个子组件在整体中所占的百分比。它们的总和应该是1，否则页面会出现没有填满的情况。

如果把columnWidth的值写错了，写成了大于1的值（有些人开始时可能不会发现前面的小数点，以为值都是整数），ColumnLayout也不会报错，但是在布局时会显得混乱。所以，当出现类似问题时，请先检查一下这里的配置是否正确。

刚才的示例较简单，下面来看一个稍微复杂一点的例子。

有时我们希望保持某一列的宽度不变，当调整页面时，只让其他列发生改变，这样来保证某一列中的布局不发生改变。在ColumnLayout中可以单独为这一列赋予宽度值，其他的列再使用columnWidth来分剩下的宽度，实现过程如代码清单8-13所示。

代码清单8-13 使用columnWidth平分剩余部分的宽度

```
var viewport = new Ext.Viewport({
    layout:'column',
    items: [{
        title: 'Column 1',
        width: 120
    },{
        title: 'Column 2',
        columnWidth: .7
    },{
        title: 'Column 3',
        columnWidth: .3
    }]
});
```

修改后，Column1的宽度定为120（像素），Column2和Column3划分剩下的宽度。Column2占剩下宽度的70%，Column3占剩下宽度的30%。以后无论页面如何改变，Column1的宽度都不会变，Column2和Column3按照70%和30%的比例进行改变。

ColumnLayout的分列规则其实很简单，一共分成两步：第一步，遍历所有items中的子组件，跳过所有设置了width值或没有设置width值而是默认使用auto的部分，同时从总宽度中减掉这

些部分占用的宽度；第二步，使用columnWidth属性的值计算出每一列的宽度，columnWidth属性的值是一个小数，表示每一列占总宽度的百分比，ColumnLayout以此来确定每一列所占的具体宽度。完成这两步操作后，最终结果就会显示到页面中。

需要特别注意的是，columnWidth的总和应该等于1，否则无法实现预期的结果。

该示例在08.layout/08-02.html中。

8.9 表格状的布局 TableLayout

先看一下表格状的布局的示例效果图，如图8-32所示。

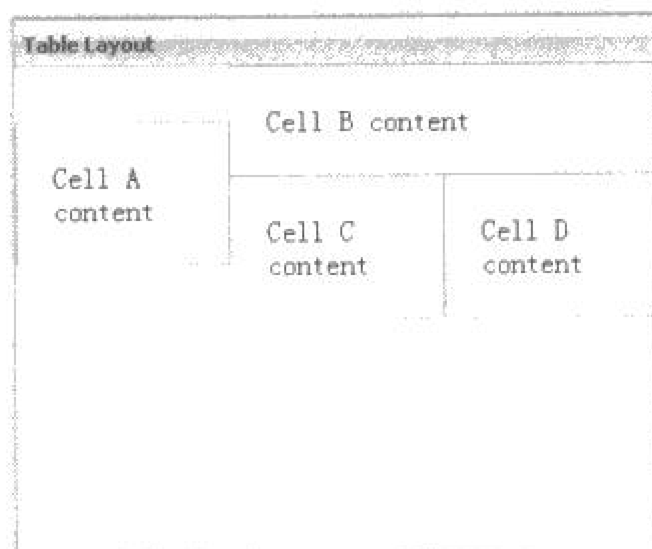


图8-32　表格布局

实现过程如代码清单8-14（见09.html）所示。

代码清单8-14　利用TableLayout进行表格布局

```
var viewport = new Ext.Viewport({
    layout:'fit',
    items:[{
        title: 'Table Layout',
        layout:'table',
        defaults: {
            bodyStyle:'padding:20px'
        },
        layoutConfig: {
            columns: 3
        },
        items: [{
            html: '<p>Cell A content</p>',
            rowspan: 2
        },{
            html: '<p>Cell B content</p>',
            colspan: 2
        },{
            html: '<p>Cell C content</p>'
        },{
```

```
                html: '<p>Cell D content</p>'
            }]
        }]
    });
```

其中，layoutConfig中定义了一共要分几列，然后在items中定义了各个子面板，使用rowspan和colspan设置了如何进行行和列的合并。它遵循从左到右、从上到下的顺序。

实际上，如果使用FireFox的Firebug查看生成的代码，可以看到生成的就是table、tr、td等标签。TableLayout为我们提供了一种将传统表格布局方式与EXT布局方式相结合的方法。

为此，TableLayout还提供了一系列配置参数对表（table）相关的标签进行配置，如表8-3所示。

表8-3　TableLayout的配置参数

参数名称	描　述
Rowspan	合并的行数
Colspan	合并的列数
cellId	某个单元格的 id
cellCls	某个单元格的 CSS 样式

TableLayout本身并没有什么特殊功能，它所实现的效果其实完全可以用BorderLayout和ColumnLayout等布局方式代替，而且这些布局方式更灵活。不过有些人更喜欢用表格布局，认为它更简单和方便。

8.10　与布局相关的其他知识

下面将介绍一些与布局相关的知识，这将有助于我们更好地理解EXT中组件和布局的应用。

8.10.1　超类 Ext.Container 的公共配置与 xtype 的概念

Ext.Container是所有可布局组件的超类，只要是继承了Ext.Container的子类就可以对自身进行布局。上面的示例中使用的都是Ext.Container的子类，那么这个超类到底为我们提供了什么公共配置呢？

最常用也最主要的就是参数layout和items，它们分别用来设置使用的布局方式和包含的子组件。这两个参数我们在前面的示例中都讲过了，只要进行布局，就必然会用到这两个参数。与这两个参数对应的还有另外两个参数，它们在需要进行特殊配置时非常有用：layoutConfig用来为布局提供特定的配置参数，实例化过程中，当前类会把自身的layoutConfig参数赋予layout对象并进行配置；activeItem则表示当前显示哪一个子组件，这在AccordionLayout和CardLayout等每次只能显示一个子组件的布局方式中非常有用。在其他布局方式中，因为所有子组件都会显示出来，所以基本上不会用到。

要重点提及的一个参数是defaultType，在子组件没有指定xtype参数时，就会使用上级组件中设置的defaultType来作为自身的xtype。默认情况下是defaultType:'panel'，也就是在items中创建的每个子组件都是Ext.Panel的实例。如果需要使用其他类型的组件，我们也无

需重新建一个对应类型的实例，只要把xtype替换成对应类型的值即可。这个功能在表单部分中可以经常看到，通过xtype可以任意指定使用Textfield、Datefield或Textarea，非常方便。

在EXT中，xtype是一大特性，{xtype:'grid'}与new Ext.grid.GridPanel()是等价的，我们可以在items中编写如下的代码：

```
items:[{
xtype:'grid',
    store: store,
    columns: columns
}]
```

也可以编写如下的代码：

```
items:[new Ext.grid.GridPanel({
    store: store,
    columns: columns
})]
```

进行整体布局时，使用xtype更方便，结构也更清晰，而使用创建实例的方式更适用于需要对某一部分进行详细配置的情况。

xtype是通过Ext.reg('grid',Ext.grid.GridPanel);的方式注册到EXT的ComponentMgr中的，在API文档中都没有提及，可以参考EXT发布包中source目录下的源代码，一般都放在最后一行，直接搜索“Ext.reg”也可以找到。

8.10.2 layout 的超类 Ext.layout.ContainerLayout

上面我们介绍了EXT提供的布局类，下面将讲解它们的超类Ext.layout.ContainerLayout，它里面只设置了所有布局类需要的一些配置，本身并没有包含布局的功能，甚至EXT中不建议使用这个类进行布局。下面我们使用Ext.layout.ContainerLayout实现一个布局示例，如代码清单8-15（见10-02.html）所示。

代码清单8-15 使用Ext.layout.ContainerLayout实现布局

```
var viewport = new Ext.Viewport({
    layout:'auto',
    items:[{
        html: 'panel 1',
    },{
        html: 'panel 2',
    },{
        html: 'panel 3',
    },{
        html: 'panel 4',
    }]
});
```

注意，Ext.layout.ContainerLayout与layout对应的关键字是auto，它意味着这个布局尚未实现任何功能，如图8-33所示。

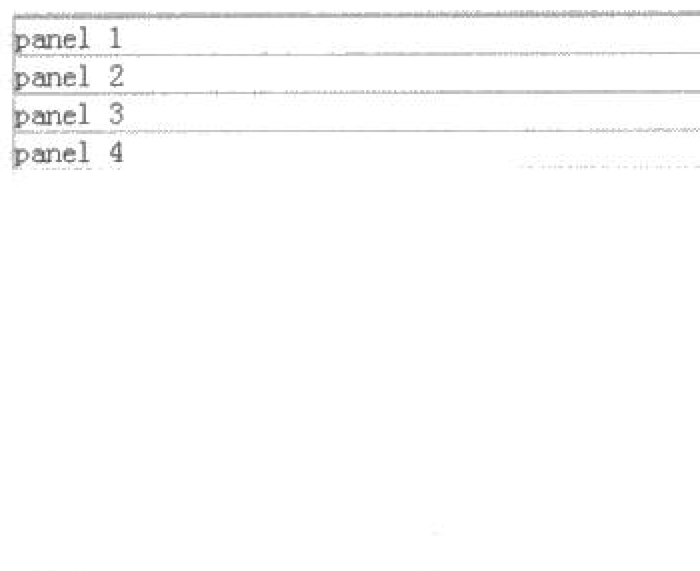


图8-33 使用ContainerLayout实现的布局

图8-33中4个面板并排放在一起，没有任何布局效果的修饰。而Ext.layout.ContainerLayout中的配置也只有两个，分别是用来指定CSS样式的extraCls和用来在渲染后隐藏所有子组件的renderHidden。它们都不常用，实际使用时完全可以把Ext.layout.ContainerLayout忽略掉。

8.10.3 不指定任何布局时会发生的情况

如果在使用Ext.Container和它的子类时不指定任何布局方式，那么会不会因为没有设置布局而出现问题呢？为此，我们先来看看表8-4中的布局类型。

表8-4 组件默认使用的布局类型

组 件	默认布局
Ext.Container	Ext.layout.ContainerLayout
Ext.Viewport	Ext.layout.ContainerLayout
Ext.Panel	Ext.layout.ContainerLayout
Ext.TabPanel	Ext.layout.CardLayout
Ext.Tip	Ext.layout.ContainerLayout
Ext.Window	Ext.layout.ContainerLayout
Ext.form.FieldSet	Ext.layout.FormPanel
Ext.form.FormPanel	Ext.layout.FormPanel
Ext.tree.TreePanel	Ext.layout.ContainerLayout
Ext.gird.GridPanel	Ext.layout.ContainerLayout
Ext.grid.EditorGridPanel	Ext.layout.ContainerLayout
Ext.grid.PropertyPanel	Ext.layout.ContainerLayout

在默认情况下，Ext.Container会在没有设置布局时创建Ext.layout.ConainterLayout，这样我们就不用担心因为忘记设置布局而出现问题了。当希望指定特定布局时，只要设置到参数里就会生效。

当然，也不是所有的子类都使用默认的Ext.layout.ConainterLayout。比如Ext.TabPanel

默认使用Ext.layout.CardLayout，Ext.form.FieldSet和Ext.form.FormPanel使用的则是Ext.layout.FormLayout。其中Ext.TabPanel更特殊，为了实现它内部的效果，它强制使用CardLayout，我们不能简单使用传递参数的方式直接修改它的布局方式，其实这是一种保护内部实现不遭到破坏的方法。

8.10.4 使用 Viewport 对整个页面进行布局

上面的示例中，我们都是使用Ext.Viewport对整个页面进行统一布局。从继承树来看，Ext.Viewport是直接继承自Ext.Container的，它基本上没有修改Ext.Container中的任何操作，只是在初始化阶段去获得当前页面的宽度和高度信息，并把自己的一些事件监听函数绑定到页面上，以此来实现跟踪页面大小的改变、动态调节自身大小的功能。

实际上Viewport只是一个用于整页布局的快捷工具类，但因为它跟页面关系紧密，会造成操作上的一些混淆。比如，有些人会忘记每个页面只能有一个Viewport的限制，直接把其他页面的布局复制过来，当作另一个页面中的子布局，并以为它可以在其他Viewport中正常工作，最后导致多个Viewport之间出现冲突，使整个页面变得混乱。

我们专门提供了一个Viewport冲突的示例，如图8-34所示。

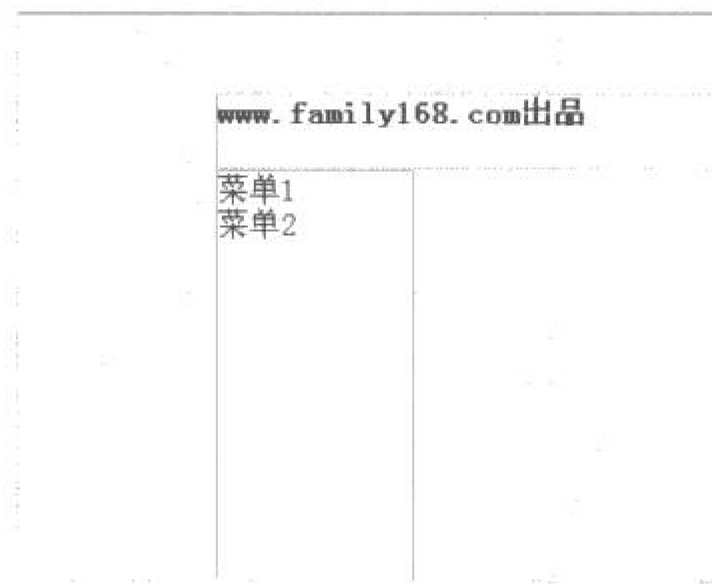


图8-34 多个Viewport冲突的情况

实现过程如代码清单8-16（见10-04.html）所示。

代码清单8-16 Viewport冲突

```
var viewport = new Ext.Viewport({
    layout: 'border',
    items: [{
        region: 'north',
        height: 40,
        html: '<h1>www.family168.com出品</h1>'
    },{
        region: 'west',
        width: 100,
```

```
            html: '<p>菜单1</p><p>菜单2</p>'
        },{
            region: 'center',
            xtype: 'viewport',
            html: '主要内容'
        }]
});
```

这里使用了BorderLayout布局方式，而center部分又使用xtype:'viewport'添加了另一个Ext.Viewport，最终造成整个页面布局失败。

这种情况下，如果想使用其他页面中写好的布局方式，就不能直接把Viewport整个复制到另一个页面里，而应该把Viewport修改为Ext.Panel，附上对应的宽度和高度信息，这样才能保证移植过去的布局可以正常工作。

8.10.5 使用嵌套实现复杂布局

嵌套布局比较复杂，下面我们来看一下Viewport里嵌套实现的布局，效果如图8-35所示。

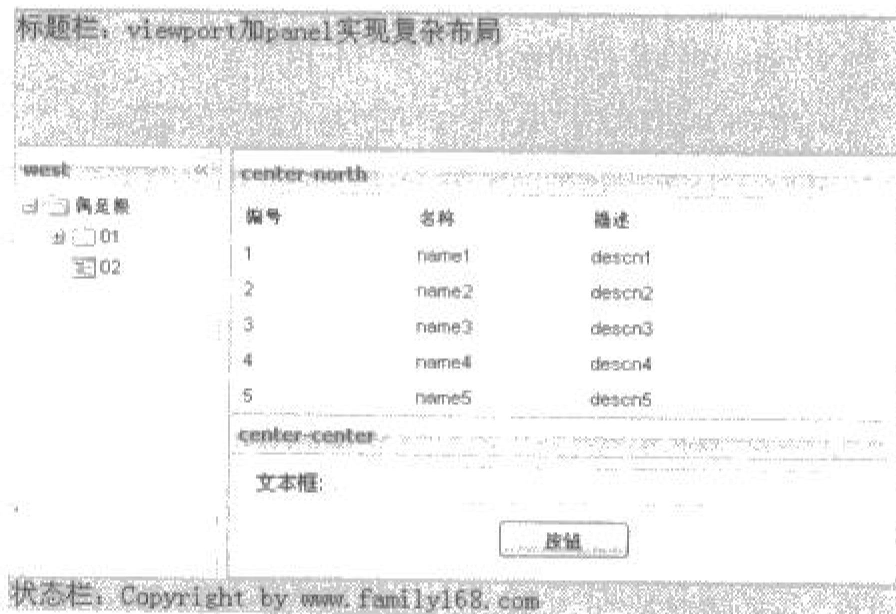


图8-35 嵌套实现的复杂布局

其实这些树形、表格和表单都是从前几章的代码中抽取出来的，使用布局组合起来之后就变成了图8-35的效果，当然也需要做一些调整。

嵌套实现复杂布局的过程如代码清单8-17所示。

代码清单8-17 嵌套实现复杂布局

```
// 布局开始
var viewport = new Ext.Viewport({
    layout:'border',
    items:[{
        region: 'north',
        contentEl: 'north-div',
        height: 80,
        bodyStyle: 'background-color:#BBCCEE;'
```

```
        },{
            region: 'south',
            contentEl: 'south-div',
            height: 20,
            bodyStyle: 'background-color:#BBCCEE;'
        },tree,{
            region: 'center',
            split: true,
            border: true,
            layout: 'border',
            items: [grid,form]
        }]
    });
    // 布局结束
```

north和south部分没有变，只是在原来的west部分直接放上了树形。center部分先设置layout:'border'使用边界布局，然后在items里分别配置表格和表单。

下面我们将分区域进行讲解。

先看一下north和south部分的代码，现在只有这两部分需要contentEl来指定HTML中的显示内容，如下面的代码所示。

```
<div id="north-div">标题栏：Viewport加Panel实现复杂布局</div>
<div id="south-div">状态栏：Copyright by www.family168.com</div>
```

布局里只设置了高度和背景颜色，可以通过bodyStyle设置整体的CSS样式。

接下来是west区域的树形。我们来看看TreePanel的定义，如代码清单8-18所示。

8

代码清单8-18　嵌套布局中的TreePanel

```
var tree = new Ext.tree.TreePanel({
    loader: new Ext.tree.TreeLoader({dataUrl: '10-05.tree.txt'}),
    title: 'west',
    region: 'west',
    split: true,
    border: true,
    collapsible: true,
    width: 120,
    minSize: 80,
    maxSize: 200
});
var root = new Ext.tree.AsyncTreeNode({text:'偶是根'});
tree.setRootNode(root);
root.expand();
```

tree里定义了很多参数，其实就是把之前写在Viewport中的用来定义大小、边界、是否可拖放、标题、是否可折叠的参数都转移了过来。

或许你会感到困惑，为什么可以把参数直接放到树形里呢？其实这个Viewport中的每个子组件都是Panel类型，而Panel包含的范围很广，有TreePanel、GridPanel和FormPanel。所以，可以放Panel的地方，也可以放它们，而且Panel的配置参数，它们也都可以用。我们之前没有指定具体用哪一种Panel，Viewport就会使用默认的Panel。现在我们指定了TreePanel，所以

Viewport原来的参数就放到TreePanel中了。

EXT的布局给我们带来了很大的方便，在任何一个Panel里设置layout:'border'就可以将它再次分成5个区域。可以在这5个区域里再次用Panel，并用region参数指定各自所处的位置。还可以在里面的Panel上设置layout，一直嵌套循环下去。

下面我们看看创建表格的那部分代码：

```
var grid = new Ext.grid.GridPanel({
    ds: ds,
    cm: cm,
    title: 'center-north',
    region: 'north'
});
```

注意，代码里多了title参数，用于设置布局上的标题，region:north说明此部分放在center区域的上方。同样，我们为表单设置了region: 'center'，代表的是Viewport的中央区域。该示例的完整代码如下所示：

```
// 表格配置开始
    var cm = new Ext.grid.ColumnModel([
        {header:'编号',dataIndex:'id'},
        {header:'名称',dataIndex:'name'},
        {header:'描述',dataIndex:'descn'}
    ]);

    var data = [
        ['1','name1','descn1'],
        ['2','name2','descn2'],
        ['3','name3','descn3'],
        ['4','name4','descn4'],
        ['5','name5','descn5']
    ];

    var ds = new Ext.data.Store({
        proxy: new Ext.data.MemoryProxy(data),
        reader: new Ext.data.ArrayReader({}, [
            {name: 'id'},
            {name: 'name'},
            {name: 'descn'}
        ])
    });
    ds.load();

    var grid = new Ext.grid.GridPanel({
        ds: ds,
        cm: cm,
        title: 'center-north',
        region: 'north'
    });
    // 表格配置结束

    // 树形配置开始
    var tree = new Ext.tree.TreePanel({
        loader: new Ext.tree.TreeLoader({dataUrl: '10-05.tree.txt'}),
```

```
    title: 'west',
    region: 'west',
    split: true,
    border: true,
    collapsible: true,
    width: 120,
    minSize: 80,
    maxSize: 200
});

var root = new Ext.tree.AsyncTreeNode({text:'偶是根'});
tree.setRootNode(root);
root.expand();
// 树形配置结束

// 表单配置开始
var form = new Ext.form.FormPanel({
    defaultType: 'textfield',
    labelAlign: 'right',
    title: 'form',
    labelWidth: 50,
    frame:true,
    width: 220,

    title: 'center-center',
    region: 'center',

    items: [{
        fieldLabel: '文本框',
        anchor: '90%'
    }],
    buttons: [{
        text: '按钮'
    }]
});
// 表单配置结束

// 布局配置开始
var viewport = new Ext.Viewport({
    layout:'border',
    items:[{
        region: 'north',
        contentEl: 'north-div',
        height: 80,
        bodyStyle: 'background-color:#BBCCEE;'
    },{
        region: 'south',
        contentEl: 'south-div',
        height: 20,
        bodyStyle: 'background-color:#BBCCEE;'
    },tree,{
        region: 'center',
        split: true,
        border: true,
```

8

```
            layout: 'border',
            items: [grid,form]
        }]
    });
```

从上面的例子中我们可以看到，结合Viewport和Panel就可以实现非常复杂而且完美的布局。

8.11　BoxLayout

在EXT 3.x之前如果希望进行分列布局只能选用ColumnLayout，实际上很少有人去用TableLayout布局方式的，但是在一些情况下只需要将几个组件平行排列，使用ColumnLayout就显得过于复杂了。

在EXT 3.x版本之后，我们可以使用BoxLayout来实现在一行中排列多个组件的功能，效果如图8-36所示。

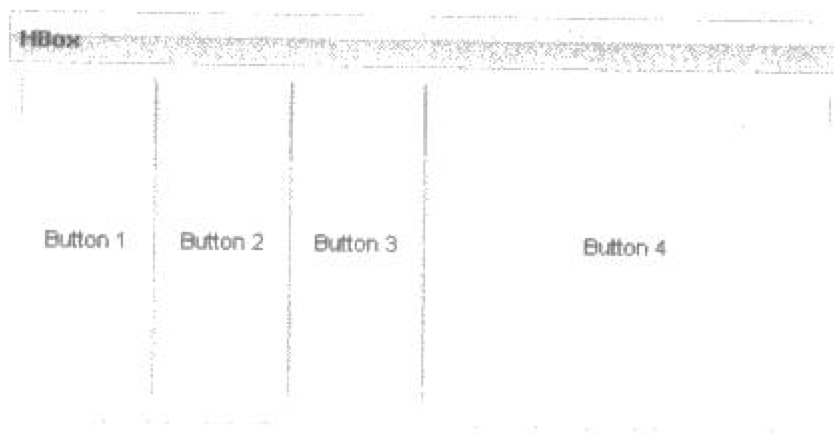


图8-36　HBox水平布局

图8-36中使用HBox布局对4个按钮进行了水平布局，实现代码如下所示：

```
var panel = new Ext.Panel({
    title: 'HBox',
    width: 400,
    height: 200,
    renderTo: 'grid',
    layout: {
        type:'hbox',
        padding:'5',
        align:'stretch'
    },
    defaults:{margins:'0 0 5 0'},
    items:[{
        xtype:'button',
        text: 'Button 1',
        flex:1
    },{
        xtype:'button',
        text: 'Button 2',
        flex:1
    },{
```

```
            xtype:'button',
            text: 'Button 3',
            flex:1
        },{
            xtype:'button',
            text: 'Button 4',
            flex:3
        }]
    });
```

与布局相关的配置放在layout属性中，我们使用type:'hbox'指定当前的Panel使用HBox布局方式，可以为其中每个组件设置flex属性，flex属性越大，对应的组件就会占据越大空间。

HBox中还支持使用align属性对布局中的组件设置统一的对齐方式，如上例中将align属性设置为'stretch'就会将Panel内部的组件高度都自动设置为充满外部容器的大小，还可以将align属性设置为'top'、'middle'、'strechmax'等值。

最新发布的EXT 3.1版本又添加了VBox布局，可以参考14.5.6节。

8.12 小结

本章主要介绍了EXT中的一大亮点——布局的应用。

主要介绍了布局的应用场景，以及哪些组件可以支持布局，并提供了布局类的继承结构图。随后详细讨论了各种布局的应用，包括FitLayout、BorderLayout、Accodion、CardLayout、AnchorLayout、AbsoluteLayout、ColumnLayout和TableLayout。

同时讨论了和布局相关的一些问题，介绍了Ext.Container和ContainerLayout的功能，还列出了组件对应的默认布局。然后介绍了如何使用Viewport对整个页面进行布局，并演示了如何使用嵌套实现复杂布局。

最后介绍了EXT 3.x中新添加的BoxLayout，介绍了如何使用HBox实现更简单的水平分列布局。

第9章

工具条和菜单

9

本章内容

- 简单菜单
- 向菜单中添加分隔线
- 多级菜单
- 高级菜单
- 工具条组件详解
- 分页工具条Ext.PagingToolbar
- 右键弹出菜单

9.1 简单菜单

菜单的种类很多，如下拉菜单、分组菜单、右键菜单，等等。右键菜单与在Windows桌面上单击右键弹出的菜单效果一样，只是样式不同。菜单上的内容包括文本、单选框、按钮等。对EXT来说，这些菜单的实现都非常简单。

我们可以在一个面板的顶端或底端放置一横排功能按钮，按下不同的按钮时会执行不同的操作。我们把这个横条称为工具条，放在工具条上的按钮称为菜单（见图9-1）。

新建　修改　删除　显示

图9-1　一个简单的工具条

这个工具条虽然有些简陋，但已经在它上面放了4个按钮。下面先来看看实现工具条的代码（见代码清单9-1），然后再来看看菜单的具体用法。

代码清单9-1　简单工具条的实现

```
// 创建工具条
var tb = new Ext.Toolbar();
tb.render('toolbar');

// 为工具条添加4个按钮
tb.add({
    text: '新建'
},{
    text: '修改'
},{
```

```
        text: '删除'
    },{
        text: '显示'
    });
```

先创建一个工具条，用工具条的render()函数把它渲染到一个DIV标签上，然后调用工具条的add函数，向工具条中添加4个按钮。在没有特别指定使用哪种类型的情况下，会默认自动转换成按钮类型。

示例在09.menu\01-01.html中。

现在单击菜单上的按钮不会有任何效果。如果想单击按钮后能执行某种操作，需要为这些按钮设置对应的事件处理函数（handler），如下面的代码所示。

```
tb.add({
    text: '新建',
    handler: function() {
        Ext.Msg.alert('提示', '新建');
    }
},{
    text: '修改',
    handler: function() {
        Ext.Msg.alert('提示', '修改');
    }
},{
    text: '删除',
    handler: function() {
        Ext.Msg.alert('提示', '删除');
    }
},{
    text: '显示',
    handler: function() {
        Ext.Msg.alert('提示', '新建');
    }
});
```

为每个按钮设置事件处理函数后，单击按钮后就会弹出相应的对话框，这样就可以为每个按钮设置功能了。虽然现在实现的功能只弹出了一个对话框，但在实际使用中，你可以在这个事件处理程序里实现任何其他的功能。

示例在09.menu\01-02.html中。

9.2 向菜单中添加分隔线

向菜单中添加分隔线的方式有如下两种。

- 使用连字符或'separator'作为参数，如下面的代码所示。

```
menuFile.add('-');
menuFile.add('separator')
```

- 用Ext.menu.Separator的实例作为参数，如下面的代码所示。

```
menuFile.add(new Ext.menu.Separator());
```

Ext.menu.Separator显示在页面上就是一条水平线，可以使用它将菜单中不同类型的菜单项分隔显示，如图9-2所示。

默认情况下，直接向菜单中添加连字符'-'就可以使用菜单分隔线了。完整代码如下所示：

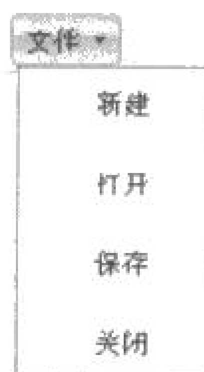


图9-2　菜单分隔条

```
var tb = new Ext.Toolbar();
    tb.render('toolbar');
    var menuFile = new Ext.menu.Menu();
    menuFile.add({text: '新建'});
    menuFile.add('-');
    menuFile.add({text: '打开'});
    menuFile.add('separator');
    menuFile.add({text: '保存'});
    menuFile.add(new Ext.menu.Separator());
    menuFile.add({text: '关闭'});

    // 为工具条添加4个按钮
    tb.add({
        text: '文件',
        menu: menuFile
    });
    tb.doLayout();
```

该示例代码在09.menu/02.html中。

9.3　多级菜单

在之前的示例中，我们只是在工具条上添加了几个按钮，如果想要这些菜单有更多的功能选项（例如图9-3中的效果），那么该怎么做呢？

如果我们想在工具条上添加更多的功能按钮，就可以像图9-3一样，把这些功能按钮放置在下拉菜单中。实现方法是先创建一个Ext.menu.Menu，然后再把它放到工具条里，如代码清单9-2所示。

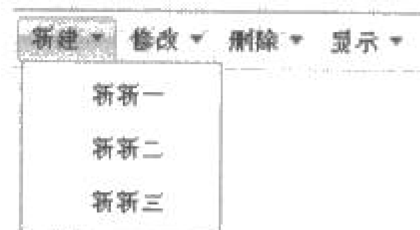


图9-3　有下拉菜单的简单工具条

代码清单9-2　下拉菜单的实现

```
var menu1 = new Ext.menu.Menu({
    items: [
        {text: '新新一'},
        {text: '新新二'},
        {text: '新新三'}
    ]
});

tb.add({
```

```
        text: '新建',
        menu: menu1
    });
```

从上面代码中可以看到，在对工具条执行add()操作的部分中，text依然对应着工具条上显示的文字，下面使用menu参数指定了用户在单击“新建”按钮时会弹出的下拉菜单。此时，工具条中显示的“新建”按钮旁边会出现一个黑色的三角形，单击它就会弹出上面定义好的菜单。

下拉菜单可以嵌套，而且嵌套的层数是任意的，我们先来看一个两层结构的菜单（见图9-4）。

实现方法很简单，只要在下拉菜单中再加上menu参数并指定对应的下级菜单即可，如代码清单9-3所示。



图9-4　一个两层结构的嵌套菜单

代码清单9-3　嵌套菜单的实现

```
var menuHistory = new Ext.menu.Menu({
    items: [
        {text: '今天'},
        {text: '昨天'},
        {text: '一周'},
        {text: '一月'},
        {text: '一年'}
    ]
});

var menuFile = new Ext.menu.Menu({
    items: [
        {text: '新建'},
        {text: '打开'},
        {text: '保存'},
        {text: '另存...'},
        '-',
        {text: '历史', menu: menuHistory},
        '-',
        {text: '关闭'}
    ]
});
```

在上面的代码中可以看出，我们可以直接使用menu参数指定在菜单的哪个部分加上子菜单。利用这些方法，我们就可以轻易实现自己想要的功能菜单了。完整代码如下所示：

```
var tb = new Ext.Toolbar();
    tb.render('toolbar');

    var menuHistory = new Ext.menu.Menu({
        items: [
            {text: '今天'},
            {text: '昨天'},
            {text: '一周'},
            {text: '一月'},
```

```
                {text: '一年'}
            ]
        });

        var menuFile = new Ext.menu.Menu({
            items: [
                {text: '新建'},
                {text: '打开'},
                {text: '保存'},
                {text: '另存...'},
                '-',
                {text: '历史', menu: menuHistory},
                '-',
                {text: '关闭'}
            ]
        });

        var menuOperator = new Ext.menu.Menu({
            items: [
                {text: '增加'},
                {text: '删除'},
                {text: '修改'}
            ]
        });

        // 为工具条添加4个按钮
        tb.add({
            text: '文件',
            menu: menuFile
        },{
            text: '操作',
            menu: menuOperator
        });
        tb.doLayout();
```

示例在09.menu/02-01.html和02-02.html中。

9.4　高级菜单

除了上面提到的最基本的菜单按钮，EXT还为我们提供了一些功能比较复杂的菜单控件，供我们直接调用。

9.4.1　多选菜单和单选菜单

这里并没有将复选框（checkbox）或单选框（radio）放到工具条上，我们看到的都是用图片实现的选择效果。

多选菜单的效果如图9-5所示，实现过程如代码清单9-4所示。



图9-5　在菜单中支持多选

代码清单9-4　在菜单中支持多选

```
var menuCheckbox = new Ext.menu.Menu({
    items: [
```

```
            new Ext.menu.CheckItem({
                text: '粗体',
                checked: true,
                checkHandler: function(item, checked) {
                    Ext.Msg.alert('多选', (checked ? '选中' : '取消') + '粗体');
                }
            }),
            new Ext.menu.CheckItem({
                text: '斜体',
                checkHandler: function(item, checked) {
                    Ext.Msg.alert('多选', (checked ? '选中' : '取消') + '斜体');
                }
            })
        ]
    });
```

我们这里使用的是Ext.menu.CheckItem，text参数表示菜单上显示的文字，checked参数表示当前菜单项是否被选中了，checkHandler用来指定选择菜单时执行的处理函数，与普通菜单不同的是，它多了一个checked参数，这个参数可以告诉我们用户执行的是选中操作，还是取消操作。

下面我们来看看单选菜单，效果如图9-6所示。

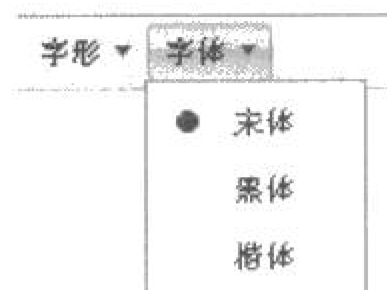


图9-6 在菜单中支持单选

实际上，在单选和多选两种情况下使用的都是Ext.menu.CheckItem，唯一不同的是group参数，如代码清单9-5所示。

代码清单9-5 在菜单中使用多选

```
    var menuRadio = new Ext.menu.Menu({
        items: [
            new Ext.menu.CheckItem({
                text: '宋体',
                group: 'font',
                checked: true,
                checkHandler: function(item, checked) {
                    Ext.Msg.alert('多选', (checked ? '选中' : '取消') + '宋体');
                }
            }),
            new Ext.menu.CheckItem({
                text: '黑体',
                group: 'font',
                checkHandler: function(item, checked) {
                    Ext.Msg.alert('多选', (checked ? '选中' : '取消') + '黑体');
                }
            }),
            new Ext.menu.CheckItem({
                text: '楷体',
                group: 'font',
                checkHandler: function(item, checked) {
                    Ext.Msg.alert('多选', (checked ? '选中' : '取消') + '楷体');
                }
            })
        ]
    });
```

我们使用group参数统一管理多个CheckItem，在单选的情况下，事件处理函数checkHandler中使用的checked参数的值一直是true。因为group中的CheckItem会在用户执行操作时自动切换本身的状态，保证每次只有一个按钮被选中。

示例在09.menu\03-01.html中。

9.4.2　日期菜单

EXT为我们提供了选择日期的菜单Ext.menu.DateMenu，它可以让我们直接把日期选择功能加入到菜单中，效果如图9-7所示。

注意，这时Ext.menu.DateMenu是一个Menu而不是MenuItem，使用时应该用menu参数将它设置成上级菜单的子菜单，如下面的代码所示。



图9-7　日期菜单

```
tb.add({
    text: '日期',
    menu: new Ext.menu.DateMenu({
        handler : function(dp, date){
            Ext.Msg.alert('选择日期', '选择的日期是 {0}.',
            date.format('Y年m月d日'));
        }
    })
});
```

注意DateMenu对应的处理函数handler，它对应的处理函数有两个参数：DatePicker和date。DatePicker表示DateMenu中对应的DatePicker对象。date表示用户选中的时间，这是一个日期对象，如果把它显示到页面上，还需要使用日期函数默认的形式转换成需要的形式。因为日期函数本身的格式是YYYY-MM-DDThh:mm:ss，而我们显示的可能是YYYY年MM月DD日，不需要中间的T或者后面时、分、秒。幸运的是，EXT内置了format函数，让我们可以更容易得到需要的格式。

该示例在09.menu\03-02.html中。

9.4.3　颜色菜单

EXT为我们提供了选择颜色的功能菜单Ext.menu.ColorMenu，效果如图9-8所示。

虽然颜色选择菜单并不常用，但它的效果十分绚丽。它的用法与日期菜单相似，也有特定的handler，让我们可以直接获得选中的颜色，如下面的代码所示。

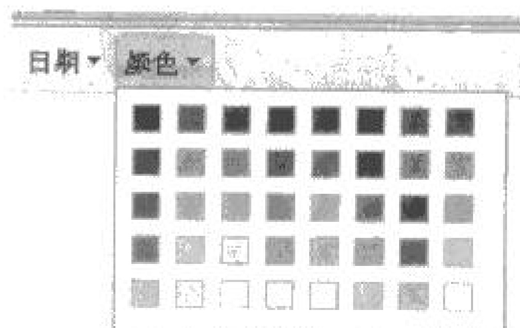


图9-8　颜色菜单

```
tb.add({
    text: '颜色',
    menu: new Ext.menu.ColorMenu({
        handler : function(cm, color){
            if (typeof color == 'string') {
                Ext.Msg.alert('选择颜色', '选择的颜色是 ' + color);
            }
```

```
            }
        })
    });
```

因为EXT本身的一些问题，某种颜色被选中时，它的handler会执行两次，第二次的参数会传入一个event对象，所以要在函数中加入typeof进行判断，以免出现问题。最后得到的颜色值是一个6位的字符串，在它前面加上#后就可以直接放到CSS里使用。

示例在09.menu\03-03.html中。

9.4.4 在菜单中添加其他组件

Ext.menu.Menu中不仅仅可以包含基本的菜单组件，也可以将Ext中的其他组件放在菜单中。下面将一个自定义的FormPanel添加到菜单中，显示效果如图9-9所示。

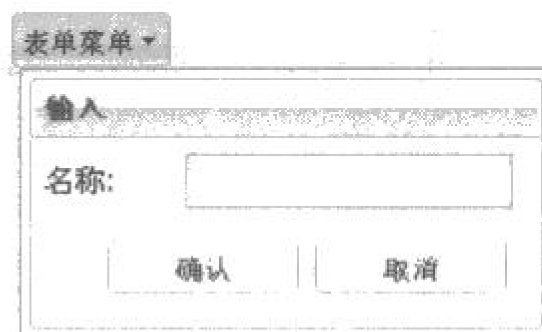


图9-9 自定义表单菜单

为了实现图9-9中的效果，我们只需要使用之前章节中的方式创建一个FormPanel，然后直接将此FormPanel添加到工具条中的menu部分即可。这样在用户点击工具条上的按钮时，就会弹出与之对应的FormPanel。

自定义表单菜单的代码如下所示：

```
Ext.onReady(function(){
    var form = new Ext.form.FormPanel({
        title: '输入',
        frame: true,
        defaultType: 'textfield',
        labelWidth: 50,
        width: 200,
        height: 100,
        items: [{
            fieldLabel: '名称',
            name: 'name'
        }],
        buttons: [{
            text: '确认'
        }, {
            text: '取消',
            function() {
            }
        }]
    });

    // 创建工具条
    var tb = new Ext.Toolbar();

    var menu = new Ext.menu.Menu({
        items: [form]
    });

    tb.add({
```

```
            text: '表单菜单',
            menu: menu
        });
        tb.render('toolbar');
        tb.doLayout();
    });
```

在将FormPanel加入菜单之后，FormPanel会自动拥有菜单的特性，如我们在菜单显示之后又使用鼠标点击了菜单之外的部分，菜单就会自动收起。如果希望实现在FormPanel上点击某些按钮就可以让菜单自动收起，可以参考下一节中Ext.menu.MenuMgr的使用方法。

示例文件在09.menu/04-04a-1.html中。

9.4.5　使用 **Ext.menu.MenuMgr** 统一管理菜单

EXT为我们提供了MenuMgr来统一管理页面上所有的菜单。每创建一个菜单都会自动注册到Ext.menu.MenuMgr中，可以使用Ext.menu.MenuMgr提供的函数对菜单进行操作。Ext.menu.MenuMgr是一个单例，我们不必创建它的实例就可以直接调用它的功能函数get()，根据id获得对应的菜单。

```
    Ext.get("showButton").on("click", function() {
        var menu = Ext.menu.MenuMgr.get("menu1");
        menu.show(tb.el);
    });
```

我们以menu的id作为参数，调用Ext.menu.MenuMgr.get()函数获得对应的menu实例后，可以直接执行各种操作，menu.show(tb.el);调用的效果是将刚刚得到的菜单显示在工具条下面，如图9-10所示。



图9-10　MenuMgr统一管理菜单

MenuMgr还提供了hideAll()函数，这个函数的作用是隐藏所有已经显示的菜单，这在需要清除页面上显示的菜单时非常有用。

```
    Ext.get('hideButton').on('click', function() {
        Ext.menu.MenuMgr.hideAll();
    });
```

对于Ext.menu.MenuMgr的统一管理功能的完整代码如下所示：

```
    var tb = new Ext.Toolbar();
        tb.render('toolbar');
        var menuHistory = new Ext.menu.Menu({
            id: 'menu1',
            allowOtherMenus: true,
            items: [
                {text: '今天'},
                {text: '昨天'},
                {text: '一周'},
                {text: '一月'},
```

```
            {text: '一年'}
        ]
    });
    var menuFile = new Ext.menu.Menu({
        id: 'menu2',
        allowOtherMenus: true,
        items: [
            {text: '新建'},
            {text: '打开'},
            {text: '保存'},
            {text: '另存...'},
            '-',
            {text: '历史', menu: menuHistory},
            '-',
            {text: '关闭'}
        ]
    });

    var menuOperator = new Ext.menu.Menu({
        id: 'menu3',
        allowOtherMenus: true,
        items: [
            {text: '增加'},
            {text: '删除'},
            {text: '修改'}
        ]
    });

    // 为工具条添加两个按钮
    tb.add({
        text: '文件',
        menu: menuFile
    },{
        text: '操作',
        menu: menuOperator
    });
    tb.doLayout();

    Ext.get("showButton").on("click", function() {
        var menu1 = Ext.menu.MenuMgr.get("menu1");
        menu1.show(tb.el);
        var menu2 = Ext.menu.MenuMgr.get("menu2");
        menu2.show(tb.el);
        var menu3 = Ext.menu.MenuMgr.get("menu3");
        menu3.show(menu2.el);
        Ext.getDoc().removeAllListeners();
    });

    Ext.get('hideButton').on('click', function() {
        Ext.menu.MenuMgr.hideAll();
    });
```

示例在09.menu/04-05.html中。

9.5 工具条组件详解

上面讲解了菜单的基本用法，下面将进一步讨论EXT中与工具条相关的组件。EXT中与工具条相关的所有组件如表9-1所示。

表9-1 工具条组件

Xtype	组 件	描 述
Toolbar	Ext.Toolbar	工具条
Tbbutton	Ext.Toolbar.Button	按钮
Tbfill	Ext.Toolbar.Fill	右对齐填充'->'
Tbitem	Ext.Toolbar.Item	工具条项目
tbseparator	Ext.Toolbar.Separator	工具条分隔符'-'
tbspacer	Ext.Toolbar.Spacer	工具条空白
tbsplit	Ext.Toolbar.SplitButton	工具条分隔按钮
tbtext	Ext.Toolbar.TextItem	工具条文本项

位于最上层的是Ext.Toolbar，通常我们都是先创建一个Ext.Toolbar工具条，然后再向里面添加各种菜单组件。

比较常用的组件有tbbutton、tbtext、tbspacer、tpseparator和tbfill。下面我们将对它们进行一一介绍。

9.5.1 Ext.Toolbar.Button

向菜单中添加按钮的方式有如下几种。

- 直接使用包含text属性的对象作为参数，如下面的代码所示。

```
tb.add({
    text: '按钮'
});
```

- 创建Ext.Button的实例作为参数，如下面的代码所示。

```
tb.add(new Ext.Button({
    text: '按钮'
}));
```

- 创建Ext.Toolbar.Button的实例作为参数，如下面的代码所示。

```
tb.add(new Ext.Toolbar.Button({
    text: '按钮'
}));
```

这3种方法都可以得到菜单按钮，其中第一种方法最简单、也最常用，之后的演示中我们就以第一种方式向菜单中添加按钮。

9.5.2 Ext.Toolbar.TextMenu

向菜单中添加文本元素的方式有如下几种。

- 使用文本字符串作为参数，如下面的代码所示。

```
tb.add('文本');
```

- 使用包含xtype: 'tbtext'属性的对象作为参数，如下面的代码所示。

```
tb.add({
    text: '文本',
    xtype: 'tbtext'
});
```

- 使用Ext.Toolbar.TextItem的实例作为参数，如下面的代码所示。

```
tb.add(new Ext.Toolbar.TextItem({
    text: '文本'
}));
```

Ext.menu.TextItem通常只用于在工具条上显示一段静态文字信息，所以我们经常使用第一种方式，直接把需要显示的信息添加到工具条中。其他两种创建方式在需要为Ext.menu.TextItem设置附加参数时才会用到。

比如需要将TextMenu对象的hideOnClick参数设置为true，如下面的代码所示。

```
tb.add({
    text: 'menu',
    menu: new Ext.menu.Menu({
        items: [{
            text: '文本',
            xtype: 'tbtext',
            hideOnClick: true
        }]
    })
});
```

这时候就需要使用指定xtype的方法，同时在参数中包含hideOnClick:true的内容。

9.5.3 Ext.Toolbar.Spacer

向菜单中添加空白元素的方式有如下几种。

- 使用空格字符作为参数，如下面的代码所示。

```
tb.add(' ');
```

- 使用包含xtype: 'tbspacer'属性的对象作为参数，如下面的代码所示。

```
tb.add({
    xtype: 'tbspacer'
});
```

- 使用Ext.Toolbar.Spacer的实例作为参数，如下面的代码所示。

```
tb.add(new Ext.Toolbar.Spacer());
```

Ext.Toolbar.Spacer是一个宽度为2 px的空白，只用来分隔两侧的工具条组件，并没有其他复杂的用法。需要时直接向工具条中添加一个空格，这就是最简单的方法。

9.5.4 Ext.Toolbar.Separator

Ext.Toolbar.Separator显示为一条竖线，用于分隔工具条组件，效果如图9-11所示。



图9-11 分隔符

向菜单中添加分隔符元素的方式有如下几种。

- 使用连字符或'separator'作为参数，如下面的代码所示。

```
tb.add('-');
tb.add('separator')
```

- 使用包含xtype：'tbseparator'属性的对象作为参数，如下面的代码所示。

```
tb.add({
    xtype: ''tbseparator '
});
```

- 使用Ext.Toolbar.Separator的实例作为参数，如下面的代码所示。

```
tb.add(new Ext.Toolbar.Separator ());
```

Ext.Toolbar.Separator与Ext.Toolbar.Spacer相似，主要用于分隔工具条中的组件，没有复杂的应用，需要时可以直接向工具条中添加连字符'-'。

9.5.5 Ext.Toolbar.Fill

Ext.Toolbar.Fill的作用是将处于它右侧的工具条组件以右对齐的方式排列在工具条右侧（效果如图9-12所示），它不会影响它左侧的组件的排列位置。



图9-12 右对齐标志

图9-12显示的是在组件中间使用Ext.Toolbar.Fill的情况，两侧的组件分别显示在工具条的两侧。如果在工具条开始处就使用Ext.Toolbar.Fill，工具条上的组件会全变成右对齐。

向菜单中添加分隔符元素的方式有如下几种。

- 使用'->'作为参数，如下面的代码所示。

```
tb.add('->');
```

- 使用包含xtype：'tbfill'属性的对象作为参数，如下面的代码所示。

```
tb.add({
    xtype: ''tbfill '
});
```

- 使用Ext.Toolbar.Fill的实例作为参数，如下面的代码所示。

```
tb.add(new Ext.Toolbar.Fill());
```

Ext.Toolbar.Fill在页面上看起来只是一段空白，它主要的功能是使工具条上的组件右对齐，并且每个工具条上只有第一个Ext.Toolbar.Fill起作用。即使在工具条上添加了多个Ext.Toolbar.Fill，也只会出现一段与右对齐对应的空白。

该示例在09.menu/05-05.html中。

9.5.6 Ext.SplitButton

一般情况下，无论是按下菜单本身还是按钮旁边的黑色小三角形按钮，都会弹出对应的子菜单。为了提升用户体验，我们可能会实现一种更便捷的方法，让菜单自动记忆上次选择的项目。下次再单击这个菜单时，不会弹出下拉菜单让用户选择，而是直接执行与上次选择的子菜单相对应的操作。

这里要讲解一下SplitButton，它也称为MenuButton。我们来看看它是如何工作的，如代码清单9-6所示。

代码清单9-6 SplitButton的工作原理

```
var menuButton = new Ext.SplitButton({
    text: '选择一个工具',
    handler: function() {
        if(menuButton.menu && !menuButton.menu.isVisible()){
            menuButton.menu.show(this.el, menuButton.menuAlign);
        }
    },
    menu: {
        items: [
            {text: '直线', handler: line},
            {text: '矩形', handler: rect},
            {text: '圆形', handler: circle}
        ]
    }
});
```


效果如图9-13所示。

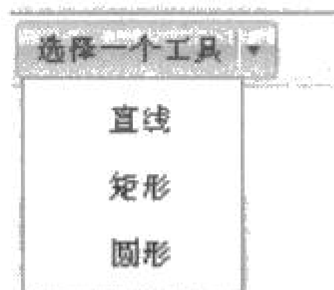


图9-13 SplitButton

下面来简单分析一下上面实现的功能。

(1) 先构造一个Ext.Toolbar.MenuButton，它本身有两种状态：单击按钮会执行handler函数；单击右边的那个小箭头会弹出菜单。

我们在这里用text指定要显示的文字，使用menu参数和对应的items定义子菜单的功能。

(2) 看看handler函数中执行了哪些操作。刚开始时，因为我们什么都没有选，所以它会直接弹出menu对应的子菜单。一旦选择了某个子菜单，就在对应的菜单项中修改MenuButton的handler，下一次再选择它时就会直接执行最后一次执行的操作。

直线对应的handler处理函数如下面代的码所示。

```
function line() {
    menuButton.handler = line;
    menuButton.setText('直线');
    Ext.Msg.alert('工具', '直线');
}
```

SplitButton的功能很有趣，这里只讲解了其中的一种用法，你可以在实际开发中去挖掘它的更多功能。完整代码如下所示：

```
var tb = new Ext.Toolbar();
    tb.render('toolbar');

    function line() {
        menuButton.handler = line;
        menuButton.setText('直线');
        Ext.Msg.alert('工具', '直线');
    }

    function rect() {
        menuButton.handler = rect;
        menuButton.setText('矩形');
        Ext.Msg.alert('工具', '矩形');
    }

    function circle() {
        menuButton.handler = circle;
        menuButton.setText('圆形');
        Ext.Msg.alert('工具', '圆形');
    }

    var menuButton = new Ext.SplitButton({
        text: '选择一个工具',
        handler: function() {
            if(menuButton.menu && !menuButton.menu.isVisible()){
                menuButton.menu.show(this.el, menuButton.menuAlign);
            }
        },
        menu: {
            items: [
                {text: '直线', handler: line},
                {text: '矩形', handler: rect},
                {text: '圆形', handler: circle}
            ]
        }
    });

    tb.add(menuButton);
    tb.doLayout();
```

示例在09.menu\05-06.html中。

9.5.7　为工具条添加 HTML 标签

我们也可以直接在工具条中加入HTML标签，包括静态文本、图片、输入框和按钮等，效果如图9-14所示。

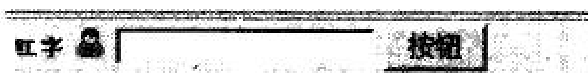


图9-14　在工具条中加入HTML

```
tb.add('<span style="color:red;font-weight:bold;">红字</span>');
tb.add('<img src="user_male.png" />');
tb.add('<input type="text">');
tb.add('<button>按钮</button>');
```

这部分内容很简单，只需要把对应的HTML片段通过add()函数加入工具条中即可。我们还可以为这些HTML片段设置CSS样式，从而控制最终的显示效果。

该示例在09.menu\05-07.html中。

9.5.8 为工具条添加输入控件

EXT还支持将表单的输入控件添加到工具条中。无论是Ext.form.TextField形式的直接填写文字内容的输入框，还是Ext.form.DateField形式的下拉选择输入框，都可以添加到工具条中，并显示，效果如图9-15所示。



图9-15 为工具条添加输入控件

实现方法如下面的代码所示：

```
tb.add('文本框: ');
tb.add(new Ext.form.TextField({
    name: 'text'
}));
tb.add('日期框: ');
tb.add(new Ext.form.DateField({
    name: 'date'
}));
```

该示例在09.menu\05-08.html中。

9.6 分页工具条 Ext.PagingToolbar

Ext.PagingToolbar继承自Ext.Toolbar，它提供了一套标准的分页组件，用来对指定的Ext.data.Store进行分页操作。完全可以把它看作一个预设了分页按钮的普通工具条。

9.6.1 Ext.PagingToolbar 的基本用法

我们在第3章讨论表格时已经接触过Ext.PagingToolbar，使用它为表格提供了分页功能。在第4章讨论ComboBox时，Ext.PagingToolbar也用来为ComboBox提供分页功能。在这些应用中，Ext.PagingToolbar都作为一个独立的组件存在，并没有与表格或ComboBox紧密结合，而是通过操作Ext.data.Store完成分页功能。

因此，只要有Ext.data.Store存在，我们就可以使用Ext.PagingToolbar完成分页功能。不过，手工显示Ext.data.Store中的数据还是有些麻烦。所以，在下面的示例中（见图9-16），我们先以Ext.grid.GridPanel作为显示数据的容器，介绍Ext.PagingToolbar的使用方式。

9

编号	名称	描述
1	name1	descn1
2	name2	descn2
3	name3	descn3

页 1 页共 2 页　　显示 1 - 3，共 5 条

图9-16　表格中的分页工具条

让我们来看一下Ext.PagingToolbar中包含的内容。首先是左侧的按钮，分别代表跳转到第1页和向前翻一页。因为目前处在第1页，所以这两个按钮都是灰色的，不可点击。

后面是提示文字和一个<input type="text">输入框，输入框中显示的是当前的页码，你可以手工输入希望跳转到的页码，敲击回车键后会跳转到对应的页面。

提示文字后面的两个按钮分别表示向后翻页和跳转到最后一页。与最左侧的两个按钮类似，如果当前是最后一页，这两个按钮也将显示成灰色，不能点击。

最后面的是一个刷新按钮，点击此按钮会执行Ext.data.Store的reload()函数，刷新Ext.data.Store的数据。在Ext.data.Store读取数据的过程中，这个图标还会有动画效果，提示数据正在读取中。

最后是右对齐的提示文字信息，显示与分页相关的一些信息。

示例中Ext.PagingToolbar定义的部分如下面的代码所示。

```
var grid = new Ext.grid.GridPanel({
    renderTo: 'grid',
    autoHeight: true,
    store: store,
    cm: cm,
    bbar: new Ext.PagingToolbar({
        pageSize: 3,
        store: store,
        displayInfo: true
    })
});
```

pageSize:3表示每页最多显示3条记录，主要用于分页操作的内部计算。

Store是关键参数，Ext.PagingToolbar在初始化时会把自己注册到store中，当store发生load事件时，会触发相关函数对Ext.PagingToolbar执行更新等操作。Ext.PaingToolbar上面的按钮输入框触发的分页操作也会直接作用在store上。

第三个参数负责显示工具条右侧的提示信息，如果为false则不会显示这些提示信息。

该示例在09.menu\06-01.html中。

9.6.2　向 **Ext.PagingToolbar** 添加按钮组件

虽然Ext.PagingToolbar上的按钮已经很多了，但有时仍然需要在上面添加一些额外的按钮组件，比如对表格进行添加、修改和删除等操作的按钮。

因为Ext.PagingToolbar继承了Ext.Toolbar，所以可以直接使用items参数为工具条设置自定义的按钮组件，如代码清单9-7所示。

代码清单9-7 向分页工具条中添加按钮组件

```
var grid = new Ext.grid.GridPanel({
    renderTo: 'grid',
    autoHeight: true,
    store: store,
    cm: cm,
    bbar: new Ext.PagingToolbar({
        pageSize: 3,
        store: store,
        displayInfo: true,
        items: ['-', {
            text: '添加',
            pressed: true
        }, ' ', {
            text: '修改',
            pressed: true
        }, ' ', {
            text: '删除',
            pressed: true
        }]
    })
});
```

添加了items参数之后，分页工具条的效果如图9-17所示。



图9-17 向分页工具条中添加按钮

如果需要处理这3个按钮的点击事件，可以分别为它们设置handler。

不仅仅是按钮，之前讨论过的工具条组件都可以加入到Ext.PagingToolbar中，它们也会依照这种顺序显示在工具条中。但请正确设置各组件的大小，否则很容易出现右侧的提示文字与自定义组件重叠的问题。

该示例在09.menu\06-02.html中。

9.7 右键弹出菜单

在EXT中，可以为用户定义一个功能菜单，在用户单击鼠标右键时，代替浏览器提供的系统功能菜单。

这种自定义的右键功能菜单也是通过Ext.menu.Menu实现的。首先定义一个多级菜单，在

它里面准备好我们需要的各种功能，如代码清单9-8所示。

代码清单9-8　右键弹出菜单

```
var menu1 = new Ext.menu.Menu({
    items: [
        {text: '新新一'},
        {text: '新新二'},
        {text: '新新三'},
    ]
});

var menu2 = new Ext.menu.Menu({
    items: [
        {text: '删删一'},
        {text: '删删二'},
        {text: '删删三'},
    ]
});

var menu3 = new Ext.menu.Menu({
    items: [
        {text: '改改一'},
        {text: '改改二'},
        {text: '改改三'},
    ]
});

var menu4 = new Ext.menu.Menu({
    items: [
        {text: '显显一'},
        {text: '显显二'},
        {text: '显显三'},
    ]
});

var contextmenu = new Ext.menu.Menu({
    items: [{
        text: '新建',
        menu: menu1
    },{
        text: '修改',
        menu: menu2
    },{
        text: '删除',
        menu: menu3
    },{
        text: '显示',
        menu: menu4
    }]
});
```

与上面示例不同的是，这里创建好的菜单并没有添加到Ext.Toolbar上，而是组合在一个名

为contextmenu的菜单下。因为没有确定渲染的位置，所以在页面上是看不到这些菜单的，效果如图9-18所示。

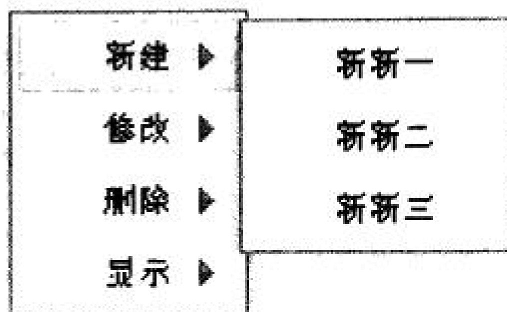


图9-18 右键弹出菜单

下面为页面添加监听事件。当用户单击右键时弹出上面定义好的功能菜单，实现过程如下所示。

```
Ext.get(document).on('contextmenu', function(e) {
    e.preventDefault();
    contextmenu.showAt(e.getXY());
});
```

首先获得document对象，这是页面中代表文档的实例，监听它的contextmenu事件，可以获得用户在浏览器上任意位置单击鼠标右键的事件。

监听函数时我们主要做两项工作，首先执行e.preventDefault()，取消浏览器对此事件的默认操作，否则，用户单击右键后还会弹出浏览器的系统菜单，同时显示两个菜单会扰乱用户的视线。其次，获得鼠标当前的坐标位置，调用contextmenu的showAt()函数，在鼠标的当前位置上显示我们定义好的功能菜单。

这里使用的是普通的Ext.menu.Menu类，大家可以在菜单上添加更多的组件，制作出功能更复杂的右键菜单。

该示例在09.menu\07.html中。

注意 EXT 3.*x*中对Button和Menu进行了大幅度加强。从EXT 3.*x*开始我们可以设置大、中、小3种形式的按钮，同时也可以在按钮和菜单中实现图文混排的效果，具体实例可以参考第14章。

9.8 小结

本章主要介绍了如何创建工具条和菜单，以及如何使用下拉菜单和分级菜单对我们需要的功能按钮进行分组显示。除此之外，本章详细讲解了与工具条相关的各种控件，包括Button、TextMenu、Spacer、Separator、Fill、SplitButton，并介绍了它们各自的配置方法和使用情况。同时，还演示了如何在工具条上增加HTML标签和表单控件。最后，本章讨论了Ext.PagingToolbar的使用方法和如何配置右键弹出菜单。

第 10 章 数据存储与传输

本章内容

- Ext.data命名空间下常用组件简介
- Ext.data.Connection
- Ext.data.Record
- Ext.data.Store
- 常用proxy
- 常用Reader
- 高级store
- EXT中的Ajax
- 关于scope和createDelegate()
- DWR与EXT整合
- localXHR支持本地使用Ajax

10.1 Ext.data 命名空间下常用组件简介

Ext.data在命名空间中定义了一系列store、reader和proxy。表格和ComboBox都是以Ext.data为媒介获取数据的，它包含异步加载、类型转换、分页等功能。Ext.data默认支持Array、JSON、XML等数据格式，可以通过Memory、HTTP、ScriptTag等方式获得这些格式的数据。如果要实现新的协议和新的数据结构，只需要扩展reader和proxy即可。DWRProxy就实现了自身的proxy和reader，让EXT可以直接从DWR获得数据。

10.2 Ext.data.Connection

Ext.data.Connection是对Ext.lib.Ajax的封装，它提供了配置使用Ajax的通用方式，在内部通过Ext.lib.Ajax实现与后台的异步调用。与底层的Ext.lib.Ajax相比，Ext.data.Connection提供了更简洁的配置方式，使用起来更方便。

Ext.data.Connection主要用于在Ext.data.HttpProxy和Ext.data.ScriptTagProxy中执行与后台交互的任务，它会从指定的URL获得数据，并把后台返回的数据交给HttpProxy或

ScriptTagProxy处理。Ext.data.Connection的使用方式如代码清单10-1所示。

代码清单10-1 使用Ext.data.Connection

```
var conn = new Ext.data.Connection({
    autoAbort: false,
    defaultHeaders: {
        referer: 'http://localhost:8080/'
    },
    disableCaching : false,
    extraParams : {
        name: 'name'
    },
    method : 'GET',
    timeout : 300,
    url : '02.txt'
});
```

在使用Ext.data.Connection之前，都要像上面这样创建一个新的Ext.Connection实例。可以在构造方法里配置对应的参数，比如autoAbort表示链接是否会自动断开、defaultHeaders参数表示请求的默认首部信息、disableCaching参数表示请求是否会禁用缓存、extraParams参数代表请求的额外参数、method参数表示请求方法、timeout参数表示连接的超时时间、url参数表示请求访问的网址等。

在创建了conn之后，可以调用request()函数发送请求，处理返回的结果，如下面的代码所示：

```
conn.request({
    success: function(response) {
        Ext.Msg.alert('info', response.responseText);
    },
    failure: function() {
        Ext.Msg.alert('warn', 'failure');
    }
});
```

Request()函数中可以设置success和failure两个回调函数，分别在请求成功和请求失败时调用。请求成功时，success函数的参数就是后台返回的信息。

下面来看一下request函数中的其他参数。

- url:String：请求url。
- params:Object/String/Function：请求传递的参数。
- method:String：请求方法，通常为GET或POST。
- callback:Function：请求完成后的回调函数，无论是成功还是失败，都会执行。
- success:Function：请求成功时的回调函数。
- failure:Function：请求失败时的回调函数
- scope:Object：回调函数的作用域。
- form:Object/String：绑定的表单。
- isUpload:Boolean：是否执行文件上传。

- headers:Object：请求首部信息。
- xmlData:Object：XML文档对象，可以通过URL附加参数的方式发起请求。
- disableCaching:Boolean：是否禁用缓存，默认为禁用。

Ext.data.Connection还提供了abort([Number transactionId])函数，当同时有多个请求发生时，根据指定的事务id放弃其中的某一个请求。如果不指定事务id，就会放弃最后一个请求。isLoading([Number transactionId])函数的用法与abort()类似，可以根据事务id判断对应的请求是否完成。如果未指定事务id，就判断最后一个请求是否完成。

该示例在10.store/02.html中。

10.3 Ext.data.Record

Ext.data.Record就是一个设定了内部数据类型的对象，它是Ext.data.Store的最基本组成部分。如果把Ext.data.Store看作是一张二维表，那么它的每一行就对应一个Ext.data.Record实例。

Ext.data.Record的主要功能是保存数据，并且在内部数据发生改变时记录修改的状态，它还可以保留修改之前的原始值。

使用Ext.data.Record时通常都是由create()函数开始的，首先用create()函数创建一个自定义的Record类型，如下面的代码所示：

```
var PersonRecord = Ext.data.Record.create([
    {name: 'name', type: 'string'},
    {name: 'sex', type: 'int'}
]);
```

PersonRecord就是我们定义的新类型，包含字符串类型的name和整数类型的sex两个属性，然后使用new关键字创建PersonRecord的实例，如下面的代码所示：

```
var boy = new PersonRecord({
    name: 'boy',
    sex: 0
});
```

创建对象时，可以直接通过构造方法为对象赋予初始值，将'boy'赋值给name，0赋值给sex。

现在，我们得到了PersonRecord的实例boy。如何才能得到它的属性呢？以下3种方式都可以获得boy中name属性的数据，如下面的代码所示：

```
alert(boy.data.name);
alert(boy.data['name']);
alert(boy.get('name'));
```

这里涉及Ext.data.Record的data属性，这是定义在Ext.data.Record中的一个公共属性，用于保存当前record对象的所有数据。它是一个JSON对象，可以直接从它里面获得需要的数据。可以通过Ext.data.Record的get()函数方便地从data属性中获得指定的属性值。

如果需要修改boy中的数据，请不要使用以下方式直接操作data，如下面的代码所示：

```
boy.data.name = 'boy name';
boy.data['name'] = 'boy name';
```

而应该使用set()函数，如下面的代码所示：

```
boy.set('name', 'body name');
```

set()函数会判断属性值是否发生了改变，如果改变了，就要将当前对象的dirty属性设置为true，并将修改之前的原始值放入modified对象中，供其他函数使用。如果直接操作data中的值，record就无法记录属性数据的修改情况。

在Record的属性数据被修改后，可以执行如下几种操作。

- commit()（提交）：这个函数的效果是设置dirty为false，并删除modified中保存的原始数据。
- reject()（撤销）：这个函数的效果是将data中已经修改了的属性值都恢复成modified中保存的原始数据，然后将dirty设置为false，并删除保存原始数据的modified对象。
- getChanges()（获得修改的部分）：这个函数会把data中经过修改的属性和数据放在一个JSON对象里并返回。例如上例中，getChanges()返回的结果是{name:'body name'}。
- 还可以调用isModified()判断当前record中的数据是否被修改。

  Ext.data.Record还提供了用于复制record实例的函数copy()。

  ```
  var copyBoy = boy.copy();
  ```

这样就得到了boy的一个副本，它里面包含了boy的data数据，但copy()函数不会复制dirty和modified等额外的属性值。

Ext.data.Record中其他的参数大多与Ext.data.Store有关，请参考与Ext.data.Store相关的讨论。

该示例在10.store/03.html中。

10.4 Ext.data.Store

Ext.data.Store是EXT中用来进行数据交换和数据交互的标准中间件，无论是表格还是ComboBox，都是通过它实现数据读取、类型转换、排序分页和搜索等操作的。

Ext.data.Store中有一个Ext.data.Record数组，所有数据都存放在这些Ext.data.Record实例中，为后面的读取和修改操作做准备。

10.4.1 基本应用

在使用之前，首先要创建一个Ext.data.Store的实例，如下面的代码所示：

```
var data = [
    ['boy', 0],
    ['girl', 1]
];

var store = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
```

```
        reader: new Ext.data.ArrayReader({}, PersonRecord)
    });
    store.load();
```

每个store最少需要两个组件的支持，分别是proxy和reader，proxy用于从某个途径读取原始数据，reader用于将原始数据转换成Record实例。

这里使用的是Ext.data.MemoryProxy和Ext.data.ArrayReader，将data数组中的数据转换成对应的几个PersonRecord实例，然后放入store中。store创建完毕之后，执行store.load()实现这个转换过程。

经过转换之后，store里的数据就可以提供给表格或ComboBox使用了，这就是Ext.data.Store的最基本用法。代码如下所示：

```
var grid = new Ext.grid.GridPanel({
    store: store,
    columns: [
        {header: 'name', dataIndex: 'name'},
        {header: 'sex', dataInex: 'sex'}
    ],
    autoHeight: true,
    renderTo: 'grid'
});
```

该示例在10.store/04-01.html中。

10.4.2　对数据进行排序

Ext.data.Store提供了一系列属性和函数，可以利用它们对数据进行排序操作。

可以在创建Ext.data.Store时使用sortInfo参数指定排序的字段和排序方式，如下面的代码所示：

```
var store = new Ext.data.Store({
    proxy: new Ext.data.MemoryProxy(data),
    reader: new Ext.data.ArrayReader({}, PersonRecord),
    sortInfo: {field: 'name', direction: 'DESC'}
});
```

这样，在store加载数据之后，就会自动根据name字段进行降序排列。对store使用store.setDefaultSort('name','DESC');也会达到同样效果。也可以在任何时候调用sort()函数，比如store.sort('name', 'DESC');，对store中的数据进行排序。

如果我们希望获得store的排序信息，可以调用getSortState()函数，返回的是类似{field: "name", direction: " DESC"}的JSON对象。

与排序相关的参数还有remoteSort，这个参数是用来实现后台排序功能的。当设置为remoteSort:true时，store会在向后台请求数据时自动加入sort和dir两个参数，分别对应排序的字段和排序的方式，由后台获取并处理这两个参数，在后台对所需数据进行排序操作。remoteSort:true也会导致每次执行sort()时都要去后台重新加载数据，而不能只对本地数据进行排序。

该示例在10.store/04-02.html中。

10.4.3 从 store 中获取数据

从store中获取数据有很多种途径，可以依据不同的要求选择不同的函数。最直接的方法是根据record在store中的行号获得对应的record，得到了record就可以使用get()函数获得里面的数据了，如下面的代码所示：

```
store.getAt(0).get('name')
```

通过这种方式，可以遍历store中所有的record，依次得到它们的数据，如下面的代码所示：

```
for (var i = 0; i < store.getCount(); i++) {
    var record = store.getAt(i);
    alert(record.get('name'));
}
```

Store.getCount()返回的是store中的所有数据记录，然后使用for循环遍历整个store，从而得到每条记录。

除了使用getCount()的方法外，还可以使用each()函数，如下面的代码所示：

```
store.each(function(record) {
    alert(record.get('name'));
});
```

each()可以接受一个函数作为参数，遍历内部record，并将每个record作为参数传递给function()处理。如果想停止遍历，可以让function()返回false。

也可以使用getRange()函数连续获得多个record，只需要指定开始和结束位置的索引值，如下面的代码所示：

```
var records = store.getRange(0, 1);
for (var i = 0; i < records.length; i++) {
    var record = records[i];
    alert(record.get('name'));
}
```

如果确实不知道record的id，也可以根据record本身的id从store中获得对应的record，如下面的代码所示：

```
store.getById(1001).get('name')
```

EXT还提供了函数find()和findBy()，可以利用它们对store中的数据进行搜索，如下面的代码所示：

```
find( String property, String/RegExp value, [Number startIndex], [Boolean anyMatch],
[Boolean caseSensitive] )
```

在这5个参数中，只有前两个是必须的。第一个参数property代表搜索的字段名；第二个参数value是匹配用字符串或正则表达式；第三个参数startIndex表示从第几行开始搜索，第四个参数anyMatch为true时，不必从头开始匹配；第五个参数caseSensitive为true时，会区分大小写。

如下面的代码所示：

```
var index = store.find('name','g');
alert(store.getAt(index).get('name'));
```

与find()函数对应的findBy()函数的定义格式如下：

```
findBy( Function fn, [Object scope], [Number startIndex] ) : Number
```

findBy()函数允许用户使用自定义函数对内部数据进行搜索。fn返回true时，表示查找成功，于是停止遍历并返回行号。fn返回false时，表示查找失败（即未找到），继续遍历，如下面的代码所示：

```
index = store.findBy(function(record, id) {
    return record.get('name') == 'girl' && record.get('sex') == 1;
});
alert(store.getAt(index).get('name'));
```

使用findBy()函数可以同时判断record中的多个字段，在函数中实现复杂逻辑。

我们还可以使用query和queryBy函数对store中的数据进行查询。与find和findBy不同的是，query和queryBy返回的是一个MixCollection对象，里面包含了搜索得到的数据，如下面的代码所示：

```
alert(store.query('name', 'boy'));
alert(store.queryBy(function(record) {
        return record.get('name') == 'girl' && record.get('sex') == 1;
}));
```

该示例在10.store/04-01.html中。

10.4.4　更新 store 中的数据

可以使用add(Ext.data.Record[] records)向store末尾添加一个或多个record，使用的参数可以是一个record实例，如下面的代码所示：

```
store.add(new PersonRecord({
    name: 'other',
    sex: 0
}));
```

add()也可以添加一个record数组，如下面的代码所示：

```
store.add([new PersonRecord({
    name: 'other1',
    sex: 0
}), new PersonRecord({
    name: 'other2',
    sex: 0
})]);
```

add()函数每次都会将新数据添加到store的末尾，这就有可能破坏store原有的排序方式。如果希望根据store原来的排序方式将新数据插入到对应的位置，可以使用addSorted()函数。它会在添加新数据之后立即对store进行排序，这样就可以保证store中的数据有序地显示，如

下面的代码所示：

```
store.addSorted(new PersonRecord({
    name: 'lili',
    sex: 1
}));
```

store会根据排序信息查找这条record应该插入的索引位置，然后根据得到的索引位置插入数据，从而实现对整体进行排序。这个函数需要预先为store设置本地排序，否则会不起作用。

如果希望自己指定数据插入的索引位置，可以使用insert()函数。它的第一个参数表示插入数据的索引位置，可以使用record实例或record实例的数组作为参数，插入之后，后面的数据自动后移，如下面的代码所示：

```
store.insert(3, new PersonRecord({
    name: 'other',
    sex: 0
}));

store.insert(3, [new PersonRecord({
    name: 'other1',
    sex: 0
}), new PersonRecord({
    name: 'other2',
    sex: 0
})]);
```

删除操作可以使用remove()和removeAll()函数，它们分别可以删除指定的record和清空整个store中的数据，如下面的代码所示：

```
store.remove(store.getAt(0));
store.removeAll();
```

store中没有专门提供修改某一行record的操作，我们需要先从store中获取一个record。对这个record内部数据的修改会直接反映到store上，如下面的代码所示：

```
store.getAt(0).set('name', 'xxxx');
```

修改record的内部数据之后有两种选择：执行rejectChanges()撤销所有修改，将修改过的record恢复到原来的状态；执行commitChanges()提交数据修改。在执行撤销和提交操作之前，可以使用getModifiedRecords()获得store中修改过的record数组。

与修改数据相关的参数是pruneModifiedRecords，如果将它设置为true，那么在每次执行删除或reload操作时，都会清空所有修改。这样，在每次执行删除或reload操作之后，getModifiedRecords()返回的就是一个空数组，否则仍然会得到上次修改过的record记录。

10.4.5 加载及显示数据

store创建好后，需要调用load()函数加载数据，加载成功后才能对store中的数据进行操作。load()调用的完整过程如下面的代码所示：

```
store.load({
```

```
        params: {start:0,limit:20},
        callback: function(records, options, success){
            Ext.Msg.alert('info', '加载完毕');
        },
        scope: store,
        add: true
});
```

- params是在store加载时发送的附加参数。
- callback是加载完毕时执行的回调函数，它包含3个参数：records表示获得的数据，options表示执行load()时传递的参数，success表示是否加载成功。
- scope用来指定回调函数执行时的作用域。
- add为true时，load()得到的数据会添加在原来的store数据的末尾，否则会先清除之前的数据，再将得到的数据添加到store中。

一般来说，为了对store中的数据进行初始化，load()函数只需要执行一次。如果用params参数指定了需要使用的参数，以后再次执行reload()重新加载数据时，store会自动使用上次load()中包含的params参数内容。

如果有一些需要固定传递的参数，也可以使用baseParams参数执行，它是一个JSON对象，里面的数据会作为参数发送给后台处理，如下面的代码所示：

```
store.baseParams.start = 0;
store.baseParams.limit = 20;
```

为store加载数据之后，有时不需要把所有数据都显示出来，这时可以使用函数filter和filterBy对store中的数据进行过滤，只显示符合条件的部分，如下面的代码所示：

```
filter( String field, String/RegExp value, [Boolean anyMatch],
[Boolean caseSensitive] ) : void
```

filter()函数的用法与之前谈到的find()相似，如下面的代码所示：

```
store.filter('name', 'boy');
```

对应的filterBy()与findBy()类似，也可以在自定义的函数中实现各种复杂判断，如下面的代码所示：

```
store.filterBy(function(record) {
    return record.get('name') == 'girl' && record.get('sex') == 1;
});
```

如果想取消过滤并显示所有数据，那么可以调用clearFilter()函数，如下面的代码所示：

```
store.clearFilter();
```

如果想知道store上是否设置了过滤器，可以通过isFiltered()函数进行判断。

10.4.6　其他功能

除了上面提到的数据获取、排序、更新、显示等功能外，store还提供了其他一些功能函数。

```
collect( String dataIndex, [Boolean allowNull], [Boolean bypassFilter] ) : Array
```

collect函数获得指定的dataIndex对应的那一列的数据。当allowNull参数为true时，返回的结果中可能会包含null、undefined或空字符串，否则collect函数会自动将这些空数据过滤掉。当bypassFilter参数为true时，collect的结果不会受查询条件的影响，无论查询条件是什么都会忽略掉，返回的信息是所有的数据，如下面的代码所示：

```
alert(store.collect('name'));
```

这样会获得所有name列的值，示例中返回的是包含了'boy'和'girl'的数组。

getTotalCount()用于在翻页时获得后台传递过来的数据总数。如果没有设置翻页，getTotalCount()的结果与getCount()相同，都是返回当前的数据总数，如下面的代码所示：

```
alert(store.getTotalCount());
```

indexOf(Ext.data.Record record)和indexOfId(String id)函数根据record或record的id获得record对应的行号，如下面的代码所示：

```
alert(store.indexOf(store.getAt(1)));
alert(store.indexOfId(1001));
```

loadData(object data, [Boolean append])从本地JavaScript变量中读取数据，append为true时，就会将读取的新数据附加到store中原有的数据后面，否则就会执行整体更新，如下面的代码所示：

```
store.loadData(data, true);
```

Sum(String property, Number start, Number end):Number用于计算某一个列从start到end的总和，如下面的代码所示：

```
alert(store.sum('sex'));
```

如果省略参数start和end，就计算全部数据的总和。

store还提供了一系列事件（见表10-1），让我们可以为对应操作设定操作函数。

10

表10-1 store提供的事件

事件名	参数
add	(Store this, Ext.data.Record[] records, Number index)
beforelaod	(Store this, Object options)
clear	(Store this)
datachanged	(Store this)
load	(Store this, Ext.data.Record[] records, Object options)
loadexception	()
metachange	(Store this, Object meta.)
remove	(Store this, Ext.data.Record record, Number index)
update	(Store this, Ext.data.Record record, String operation)

至此，store和record等组件已经讲解完毕。下面主要讨论一下常用的proxy和reader组件。

10.5 常用 proxy

本节将介绍一些常用的proxy，这些proxy的作用是通过内存，HTTP等不同的媒介获取原始数据，然后将获取的数据交给对应的读取器进行处理。

10.5.1 MemoryProxy

MemoryProxy只能从JavaScript对象获得数据，可以直接把数组，或JSON和XML格式的数据交给它处理，如下面的代码所示：

```
var proxy = new Ext.data.MemoryProxy([
    ['id1','name1','descn1'],
    ['id2','name2','descn2']
]);
```

10.5.2 HttpProxy

HttpProxy使用HTTP协议，通过Ajax去后台取数据，构造它时需要设置url:'xxx.jsp'参数。这里的url可以替换成任何一个合法的网址，这样HttpProxy才知道去哪里获取数据，如下面的代码所示：

```
var proxy = new Ext.data.HttpProxy({url:'xxx.jsp'});
```

后台需要返回EXT所需要的JSON格式的数据。下面的内容就是后台使用JSP的一个范例，如下面的代码所示：

```
response.setContentType("application/x-json");
Writer out = response.getWriter();
out.print("[" +
        "['id1','name1','descn1']" +
        "['id2','name2','descn2']" +
    "]");
```

请注意，这里的HttpProxy不支持跨域，它只能从同一域中获得数据。如果想跨域，请参考下面的ScriptTagProxy。

10.5.3 ScriptTagProxy

ScriptTagProxy的用法几乎和HttpProxy一样，如下面的代码所示：

```
var proxy = new Ext.data.ScriptTagProxy({url:'xxx.jsp'});
```

从这里也看不出来它是如何支持跨域的，我们还需要在后台进行相应的处理，如下面的代码所示：

```
String cb = request.getParameter("callback");
response.setContentType("text/javascript");
Writer out = response.getWriter();
out.write(cb + "(");
out.print("[" +
```

```
            "['id1','name1','descn1']" +
            "['id2','name2','descn2']" +
        "]");
    out.write(");");
```

其中的关键就在于从请求中获得的callback参数，这个参数叫做回调函数。ScriptTagProxy会在当前的HTML页面里添加一个<script type="text/javascript"src="xxx.jsp"> </script>标签，然后把后台返回的内容添加到这个标签中，这样就可以解决跨域访问数据的问题。为了让后台返回的内容可以在动态生成的标签中运行，EXT会生成一个名为callback的回调函数，并把回调函数的名称传递给后台，由后台生成callback(data)形式的响应内容，然后返回给前台自动运行。

虽然上述处理过程比较难理解，但是我们只需要了解ScriptTagProxy的用法就足够了。如果还想进一步了解ScriptTagProxy的运行过程，可以使用Firebug查看动态生成的HTML以及响应的JSON内容。

最后我们来分析一下EXT的API文档中提供的示例，这段后台代码会自动判断请求的类型，返回支持ScriptTagProxy或HttpProxy的数据，如代码清单10-2所示。

代码清单10-2 在后台同时支持HttpProxy和ScriptTagProxy

```
boolean scriptTag = false;
String cb = request.getParameter("callback");
if (cb != null) {
    scriptTag = true;
    response.setContentType("text/javascript");
} else {
    response.setContentType("application/x-json");
}
Writer out = response.getWriter();
if (scriptTag) {
    out.write(cb + "(");
}
out.print(dataBlock.toJsonString());
if (scriptTag) {
    out.write(");");
}
```

代码中通过判断请求中是否包含callback参数来决定返回何种数据类型。如果包含，就返回ScriptTagProxy需要的数据；否则，就当作HttpProxy处理。

10.6 常用 Reader

本节介绍常用的数据读取器，这些读取器的作用是将数组、JSON等格式的原始数据转换为EXT中所需要的通用数据类型。

10.6.1 ArrayReader

从proxy中读取的数据需要进行解析，这些数据转换成Record数组后才能提供给

Ext.data.Store使用。

ArrayReader的作用是从二维数组里依次读取数据，然后生成对应的Record。默认情况下是按列顺序读取数组中的数据。不过你也可以考虑用mapping指定record与原始数组对应的列号。ArrayReader的用法很简单，但缺点是不支持分页。使用二维数组的方式如下面的代码所示：

```
var data = [
    ['id1','name1','descn1'],
    ['id2','name2','descn2']
];
```

对应的ArrayReader如下面的代码所示：

```
var reader = new Ext.data.ArrayReader({
    id:1
},[
    {name:'name',mapping:1},
    {name:'descn',mapping:2},
    {name:'id',mapping:0},
]);
```

我们演示的是字段顺序不一致的情况，如果字段顺序和列顺序一致，就不用额外配置mapping。

10.6.2 JsonReader

在JavaScript中，JSON是一种非常重要的数据格式，key:value的形式比XML那种复杂的标签结构更容易理解，代码量也更小，很多人倾向于使用它作为EXT的数据交换格式。为JsonReader准备的JSON数据如下面的代码所示：

```
var data = {
    id:0,
    totalProperty:2,
    successProperty:true,
    root:[
        {id:'id1',name:'name1',descn:'descn1'},
        {id:'id2',name:'name2',descn:'descn2'}
    ]
};
```

与数组相比，JSON的最大优点就是支持分页，我们可以使用totalProperty参数表示数据的总量。successProperty参数是可选的，可以用它判断当前请求是否执行成功，进而判断是否进行数据加载。在不希望JsonReader处理响应数据时，可以把successProperty设置成false。

现在来讨论一下JsonReader，看看它是如何与上面的JSON数据对应的，如下面的代码所示：

```
var reader = new Ext.data.JsonReader({
    successProperty: "successproperty",
    totalProperty: "totalProperty",
    root: "root",
    id: "id"
```

```
}, [
    {name:'id',mapping:'id'},
    {name:'name',mapping:'name'},
    {name:'descn',mapping:'descn'}
]);
```

上例中的对应方式不够简洁，因为name和mapping部分的内容是相同的。其实这里的mapping可以省略，默认会用name参数从JSON中获得对应的数据。如果不想与JSON里的名字一样，也可以使用mapping修改。不过，mapping在这里还有其他用途，如代码清单10-3所示。

代码清单10-3 为JsonReader设置mapping进行数据映射

```
var data = {
    id:0,
    totalProperty:2,
    successProperty:true,
    root:[
        {id:'id1',name:'name1',descn:'descn1',person:{
            id:1,name:'man',sex:'male'
        }},
        {id:'id2',name:'name2',descn:'descn2',person:{
            id:2,name:'woman',sex:'female'
        }}
    ]
};
var reader = new Ext.data.JsonReader({
    successProperty: "successproperty",
    totalProperty: "totalProperty",
    root: "root",
    id: "id"
}, [
    'id','name','descn',
    {name:'person_name',mapping:'person.name'},
    {name:'person_sex',mapping:'person.sex'}
]);
```

在上面的代码中，我们使用JSON支持更复杂的嵌套结构，其中的person对象自身就拥有id、name和sex等属性。在JsonReader中可以用mapping把这些嵌套的内部属性映射出来，赋予对应的record，而其他字段都不变。

10.6.3 XmlReader

XML是非常通用的数据传输格式，XmlReader使用的XML格式的数据如代码清单10-4所示。

代码清单10-4 XmlReader使用的XML格式的数据

```
<?xml version="1.0" encoding="utf-8"?>
<dataset>
    <id>1</id>
    <totalRecords>2</totalRecords>
    <success>true</success>
    <record>
```

```
            <id>1</id>
            <name>name1</name>
            <descn>descn1</descn>
        </record>
        <record>
            <id>2</id>
            <name>name2</name>
            <descn>descn2</descn>
        </record>
</dataset>
```

这里一定要用dataset作为XML根元素。再让我们看一下如何对XmlReader进行配置，从而读取上面示例中的XML数据，如下面的代码所示：

```
var reader = new Ext.data.XmlReader({
   totalRecords: 'totalRecords',
   success: 'success'
   record: 'record',
   id: "id"
}, ['id','name','descn']);
```

XmlReader使用的参数与之前介绍的JsonReader有些不同，我们可以看到这里用到了totalRecords和record两个参数，其中totalRecords用来指定从'totalRecords'标签里获得后台数据总数，record则表示XML中放在record标签里的数据是我们需要显示的结果数据。其他两个参数success和id的含义和JsonReader中对应的参数相似，分别用来判断操作是否成功和这次返回的id。因为XML中的标签和reader里需要的名字是相同的，所以简化了配置，将[{name: 'id'},{name:'name'},{name:'descn'}]直接写成了['id','name','descn']。

因为XmlReader不能将JavaScript中的字符串自动解析成XML格式的数据，所以需要利用其他方法进行演示。参考localXHR.js中构造XML的方式，我们有了下面的解决方案，如代码清单10-5所示。

代码清单10-5　通过本地字符串构造XML对象

```
var data = "<?xml version='1.0' encoding='utf-8'?>" +
    "<dataset>" +
        "<id>1</id>" +
        "<totalRecords>2</totalRecords>" +
        "<success>true</success>" +
        "<record>" +
            "<id>1</id>" +
            "<name>name1</name>" +
            "<descn>descn1</descn>" +
        "</record>" +
        "<record>" +
            "<id>2</id>" +
            "<name>name2</name>" +
            "<descn>descn2</descn>" +
        "</record>" +
    "</dataset>";
```

```
var xdoc;

if(typeof(DOMParser) == 'undefined'){
    xdoc = new ActiveXObject("Microsoft.XMLDOM");
    xdoc.async="false";
    xdoc.loadXML(data);
}else{
    var domParser = new DOMParser();
    xdoc = domParser.parseFromString(data, 'application/xml');
    domParser = null;
}

var proxy = new Ext.data.MemoryProxy(xdoc);

var reader = new Ext.data.XmlReader({
    totalRecords: 'totalRecords',
    success: 'success',
    record: 'record',
    id: "id"
}, ['id','name','descn']);

var ds = new Ext.data.Store({
    proxy: proxy,
    reader: reader
});
```

10.7 高级 store

实际开发时，并不需要每次都对proxy、reader、store这3个对象进行配置，EXT提供了几种可选择的整合方案。

- SimpleStore = Store + MemoryProxy + ArrayReader

```
var ds = Ext.data.SimpleStore({
    data: [
            ['id1','name1','descn1'],
            ['id2','name2','descn2']
    ],
    fields: ['id','name','descn']
});
```

SimpleStore是专为简化读取本地数组而设计的，设置好MemoryProxy需要的data和ArrayReader需要的fields就可以使用了。

- JsonStore = Store + HttpProxy + JsonReader

```
var ds = Ext.data.JsonStore({
    url: 'xxx.jsp',
    root: 'root',
    fields: ['id','name','descn']
});
```

JsonStore将JsonReader和HttpProxy整合在一起，提供了一种从后台读取JSON信息的简便方法，大多数情况下可以考虑直接使用它从后台读取数据。

- Ext.data.GroupingStore可以对数据进行分组。

  Ext.data.GroupingStore继承自Ext.data.Store，它的主要功能是可以对内部的数据进行分组。可以在创建Ext.data.GroupingStore时指定根据某个字段进行分组，也可以在创建实例后调用它的groupBy()函数对内部数据重新分组，如下面的代码所示：

```
var ds = new Ext.data.GroupingStore({
    data: [
        ['id1','name1','female','descn1'],
        ['id2','name2','male','descn2'],
        ['id3','name3','female','descn3'],
        ['id4','name4','male','descn4'],
        ['id5','name5','female','descn5']
    ],
    reader: new Ext.data.ArrayReader({
        fields: ['id','name','sex','descn']
    }),
    groupField: 'sex',
    groupOnSort: true
});
```

上例中，我们使用groupField作为参数，为Ext.data.Grouping设置了分组字段，另外还设置了groupOnSort参数，这个参数可以保证只有在进行分组时才会对Ext.data.GroupingStore内部的数据进行排序。如果采用默认值，就需要手工指定sortInfo参数，从而指定默认的排序字段和排序方式，否则就会出现错误。

创建Ext.data.GroupingStore的实例之后，我们还可以调用groupBy()函数重新对数据进行分组。因为设置了groupOnSort:true，所以在重新分组时，EXT会使用分组的字段对内部数据进行排序。如果不想对数据进行分组，也可以调用clearGrouping()函数清除分组信息，如下面的代码所示：

```
ds.groupBy('id');
ds.clearGrouping();
```

该示例在10.store/07.html中。

10.8 EXT 中的 Ajax

EXT与后台交换数据时，很大程度上依赖于底层实现的Ajax。所谓底层实现，就是说很可能就是我们之前提到的 Prototype、jQuery或YUI中提供的Ajax功能。为了统一接口，EXT在它们的基础上进行了封装，让我们可以用同一种写法“游走”于各种不同的底层实现之间。

10.8.1 最容易看到的 Ext.Ajax

Ext.Ajax的基本用法如下所示：

```
Ext.Ajax.request({
    url: '07-01.txt',
    success: function(response) {
        Ext.Msg.alert('成功', response.responseText);
```

```
        },
        failure: function(response) {
            Ext.Msg.alert('失败', response.responseText);
        },
        params: { name: 'value' }
    });
```

这里调用的是Ext.Ajax的request函数，它的参数是一个JSON对象，具体如下所示。

- url参数表示将要访问的后台网址。
- success参数表示响应成功后的回调函数。

上例中我们直接从response取得返回的字符串，用Ext.Msg.alert显示出来。

- failure参数表示响应失败后的回调函数。

注意，这里的响应失败并不是指数据库操作之类的业务性失败，而是指HTTP返回404或500错误，请不要把HTTP响应错误与业务错误混淆在一起。

- params参数表示请求时发送到后台的参数，既可以使用JSON对象，也可以直接使用"name=value"形式的字符串。

上面的示例可以在10.store/08-01.html中找到。

Ext.Ajax直接继承自Ext.data.Connection，只不过它是一个单例，不需要用new创建实例，可以直接使用。在使用Ext.data.Connection前需要先创建实例，因为Ext.data.Connection是为了给Ext.data中的各种proxy提供Ajax功能，分配不同的实例更有利于分别管理。Ext.Ajax为用户提供了一个简易的调用接口，实际使用时，可以根据自己的需要进行选择。

10.8.2 Ext.lib.Ajax 是更底层的封装

其实Ext.Ajax和Ext.data.Connection的内部功能实现都是依靠Ext.lib.Ajax来完成的，在Ext.lib.Ajax下面就是各种底层库的Ajax了。

如果使用Ext.lib.Ajax实现以上的功能，就需要写成下面的形式，如下面的代码所示：

```
Ext.lib.Ajax.request(
    'POST',
    '07-01txt',
    {success: function(response){
        Ext.Msg.alert('成功', response.responseText);
    },failure: function(){
        Ext.Msg.alert('失败', response.responseText);
    }},
    'data=' + encodeURIComponent(Ext.encode({name:'value'}))
);
```

我们可以看到，使用Ext.lib.Ajax时需要传递4个参数，分别为method、url、callback和params。它们的含义与Ext.Ajax中的参数都是一一对应的，唯一没有提到过的method参数表示请求HTTP的方法，它也可以在Ext.Ajax中使用method:'POST'的方式设置。

相对于Ext.Ajax来说，Ext.lib.Ajax有如下几个缺点。

- 参数的顺序被定死了，第一个参数是method，第二个参数是url，第三个参数是回调函数callback，第四个参数是params。这样既不容易记忆，也无法省略其中某个不需要的

参数。Ext.Ajax中用JSON对象来定义参数，使用起来更灵活。

- 在params部分，Ext.lib.Ajax必须使用字符串形式，显得有些笨重。Ext.Ajax则可以在JSON对象和字符串之间随意选择，非常灵活。

比与Ext.Ajax相比，Ext.lib.Ajax的唯一优势就是它可以在EXT 1.x中使用。如果你使用的是EXT 2.0或更高的版本，那么就放心大胆地使用Ext.Ajax吧，它会带给你更多的惊喜。

该示例在10.store/08-02.html中。

10.9 关于 scope 和 createDelegate()

JavaScript中this的使用是一个由来已久的问题了。这里就不介绍它的发展历史了，只结合具体的例子，告诉大家可能会遇到什么问题，及遇到这些问题时EXT是如何解决的。在使用EXT时，最常碰到的就是使用Ajax回调函数时出现的问题，如下面的代码所示。

```
<input type="text" name="text" id="text">
<input type="button" name="button" id="button" value="button">
```

现在的HTML 页面中有一个text输入框和一个按钮。我们希望按下这个按钮之后，能用Ajax去后台读取数据，然后把后台响应的数据放到text中，实现过程如代码清单10-6所示。

代码清单10-6 Ajax中使用回调函数

```
function doSuccess(response) {
    text.dom.value = response.responseText;
}

Ext.onReady(function(){
    Ext.get('button').on('click', function(){
        var text = Ext.get('text');
        Ext.lib.Ajax.request(
            'POST',
            '08.txt',
            {success:doSuccess},
            'param=' + encodeURIComponent(text.dom.value)
        );
    });
});
```

在上面的代码中，Ajax已经用Ext.get('text')获得了text，以为后面可以直接使用，没想到回调函数success不会按照你写的顺序去执行。当然，也不会像你所想的那样使用局部变量text。实际上，如果什么都不做，只使用回调函数，你不得不再次使用Ext.get('text')重新获得元素，否则浏览器就会报text未定义的错误。

在此使用Ext.get('text')重新获取对象还比较简单，在有些情况下不容易获得需要处理的对象，我们要在发送Ajax请求之前获取回调函数中需要操作的对象，有两种方法可供选择：scope和createDelegate。

- 为Ajax设置scope。

```
function doSuccess(response) {
    this.dom.value = response.responseText;
}
Ext.lib.Ajax.request(
    'POST',
    '09.txt',
    {success:doSuccess,scope:text},
    'param=' + encodeURIComponent(text.dom.value)
);
```

在Ajax的callback参数部分添加一个scope:text，把回调函数的scope指向text，它的作用就是把doSuccess函数里的this指向text对象。然后再把doSuccess里改成this.dom.value，这样就可以了。如果想再次在回调函数里用某个对象，必须配上scope，这样就能在回调函数中使用this对它进行操作了。

- 为success添加createDelegate()。

```
function doSuccess(response) {
    this.dom.value = response.responseText;
}

Ext.lib.Ajax.request(
    'POST',
    '09.txt',
    {success:doSuccess.createDelegate(text)},
    'param=' + encodeURIComponent(text.dom.value)
);
```

createDelegate只能在function上调用，它把函数里的this强行指向需要的对象，然后我们就可以在回调函数doSuccess里直接通过this来引用createDelegate()中指定的这个对象了。它可以作为解决this问题的一个备选方案。

建议大家尽量选择scope来控制JavaScript的作用域，因为createDelegate是要对原来的函数进行封装，重新生成function对象。简单环境下，scope就够用了，倒是createDelegate还有其他功能，比如修改调用参数等。

示例在10.store/08.html中。

10

10.10 DWR 与 EXT 整合

据不完全统计，从事Ajax开发的Java程序员有一大半都使用DWR。下面我们来介绍一下如何在EXT中使用DWR与后台交互。

10.10.1 在 EXT 中直接使用 DWR

因为DWR在前台的表现形式和普通的JavaScript完全一样，所以我们不需要特地去做些什么，直接使用EXT调用DWR生成的JavaScript函数即可。以表格为例，比如现在我们要显示一个通讯录的信息，后台记录的数据有id、name、sex、email、tel、addTime和descn。编写对应的POJO，代码如下所示：

```
public class Info {
```

```
    long id;
    String name;
    int sex;
    String email;
    String tel;
    Date addTime;
    String descn;
}
```

然后编写操作POJO的manager类，代码如下所示：

```
public class InfoManager {
    private List infoList = new ArrayList();

    public List getResult() {
        return infoList;
    }
}
```

代码部分有些删减，我们只保留了其中的关键部分，就这样把这两个类配置到dwr.xml中，让前台可以对这些类进行调用。

下面是EXT与DWR交互的关键部分，我们要对JavaScript部分做如下修改，如代码清单10-7所示。

代码清单10-7 使用EXT调用DWR

```
var cm = new Ext.grid.ColumnModel([
    {header:'编号',dataIndex:'id'},
    {header:'名称',dataIndex:'name'},
    {header:'性别',dataIndex:'sex'},
    {header:'邮箱',dataIndex:'email'},
    {header:'电话',dataIndex:'tel'},
    {header:'添加时间',dataIndex:'addTime'},
    {header:'备注',dataIndex:'descn'}
]);

var store = new Ext.data.JsonStore({
    fields: ["id","name","sex",'email','tel','addTime','descn']
});

// 调用DWR取得数据
infoManager.getResult(function(data) {
    store.loadData(data);
});

var grid = new Ext.grid.GridPanel({
    renderTo: 'grid',
    store: store,
    cm: cm
});
```

注意，执行infoManager.getResult()函数时，DWR就会使用Ajax去后台取数据了，操作成功后调用我们定义的匿名回调函数。在这里我们只做一件事，那就是将返回的data直接注入

到ds中。

DWR返回的data可以被JsonStore直接读取，我们需要设置对应的fields参数，告诉JsonReader需要哪些属性。

在这里，EXT和DWR两者之间没有任何关系，将它们任何一方替换掉都可以。实际上它们只是在一起运行，并没有整合。我们给出的这个示例也是说明了一种松耦合的可能性，实际操作中完全可以使用这种方式。

10.10.2 DWRProxy

要结合使用EXT和DWR，不需要对后台程序进行任何修改，可以直接让前后台数据进行交互。不过还要考虑很多细节，比如表格分页、刷新、排序、搜索等常见的操作。EXT的官方网站上已经有人放上了DWRProxy，借助它可以让DWR和EXT连接得更加紧密。不过，需要在后台添加DWRProxy所需要的Java类，这可能不是最好的解决方案。但我们相信，通过对它的内在实现的讨论，我们可以有更多的选择和想象空间。

注意 这个DWRProxy.js一定要放在ext-base.js和ext-all.js后面，否则会出错。

我们现在就用DWRProxy来实现一个分页的示例。除了准备好插件DWRProxy.js外，还要在后台准备一个专门用于分页的封装类。因为不仅要告诉前台显示哪些数据，还要告诉前台一共有多少条数据。现在我们来重点看一下ListRange.java，如下面的代码所示：

```
public class ListRange {
    Object[] data;
    int totalSize;
}
```

其实ListRange非常简单，只有两个属性：提供数据的data和提供数据总量的totalSize。再看一下InfoManager.java。为了实现分页，我们专门编写了一个getItems方法，代码如下所示：

```
public ListRange getItems(Map conditions) {
    int start = 0;
    int pageSize = 10;
    int pageNo = (start / pageSize) + 1;

    try {
        start = Integer.parseInt(conditions.get("start").toString());
        pageSize = Integer.parseInt(conditions.get("limit").toString());
        pageNo = (start / pageSize) + 1;
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    List list = infoList.subList(start, start + pageSize);
    return new ListRange(list.toArray(), infoList.size());
}
```

getItems()的参数是Map，我们从中获得需要的参数，比如start和limit。不过HTTP里

的参数都是字符串，而我们需要的是数字，所以要对类型进行相应的转换。根据start和limit两个属性从全部数据中截取一部分，放进新建的ListRange中，然后把生成的ListRange返回给前台，于是一切都解决了。

重头戏要上演了，我们就要使用传说中的Ext.data.DWRProxy了，还有Ext.data.ListRangeReader。通过这两个扩展，EXT完全可以支持DWR的数据传输协议。实际上，这正是EXT要把数据和显示分离设计的原因，这样你只需要添加自定义的proxy和reader，不需要修改EXT的其他部分，就可以实现从特定途径获取数据的功能。后台还是DWR，所以至少在表格部分，我们可以很好地使用它们的结合，主要代码如下所示：

```
var store = new Ext.data.Store({
    proxy: new Ext.data.DWRProxy(infoManager.getItems, true),
    reader: new Ext.data.ListRangeReader({
        totalProperty: 'totalSize',
        root: 'data',
        id: 'id'
    }, info),
    remoteSort: true
});
```

与前面讲述的一样，我们修改了proxy，也修改了reader，其他地方都不需要进行修改，表格已经可以正常运行了。需要提醒的是DWRProxy的用法，其中包括两个参数：第一个是dwrCall，它把一个DWR函数放进去，它对应的是后台的getItems方法；第二个参数是pagingAndSort，这个参数控制DWR是否需要分页和排序。

ListRangeReader部分与后台的ListRange.java对应。totalProperty表示后台数据总数，我们通过它指定从ListRange中读取totalSize属性的值来作为后台数据总数。还需要指定root参数，告诉它在ListRange中的数据变量的名称为data，随后DWRProxy会从ListRange的data属性中获取数据并显示到页面上。如果不想使用我们提供的ListRange.java类，也可以自己创建一个类，只要把totalProperty和data两个属性与之对应即可。

10.10.3　DWRTreeLoader

我们现在来尝试一下让树形也支持DWR。有了前面的基础，整合DWR和树形就更简单了。在后台，我们需要树形节点对应的TreeNode.java。目前，只要id、text和leaf3项就可以了。

```
public class TreeNode {
    String id;
    String text;
    boolean leaf;
}
```

id是节点的唯一标记，知道了id就能知道是在触发哪个节点了。text是显示的标题，leaf比较重要，它用来标记这个节点是不是叶子。

这里还是用异步树，TreeNodeManager.java里的getTree()方法将获得一个节点的id作为参数，然后返回这个节点下的所有子节点。这里没有限制生成的树形的深度，你可以根据自己的需要进行设置。TreeNodeManager.java的代码如下所示：

```
public List getTree(String id) {
    List list = new ArrayList();
    String seed1 = id + 1;
    String seed2 = id + 2;
    String seed3 = id + 3;
    list.add(new TreeNode(seed1, "" + seed1, false));
    list.add(new TreeNode(seed2, "" + seed2, false));
    list.add(new TreeNode(seed3, "" + seed3, true));

    return list;
}
```

上面的代码并不复杂，它实现的效果与在Java中使用List或数组是相同的，因为返回给前台的数据都是JSON格式的。前台使用JavaScript处理返回信息的部分更简单，先引入DWRTree-Loader.js，然后把TreeLoader替换成DWRTreeLoder即可，如下面的代码所示：

```
var tree = new Ext.tree.TreePanel('tree', {
    loader: new Ext.tree.DWRTreeLoader({dataUrl: treeNodeManager.getTree})
});
```

参数依然是dataUrl，它的值treeNodeManager.getTree代表的是一个DWR函数，我们不需要对它进行深入研究，它的内部会自动处理数据之间的对应关系。DWR有时真的很方便。

10.10.4 DWRProxy 和 ComboBox

DWRProxy既然可以用在Ext.data.Store中，那么它也可以为ComboBox服务，如代码清单10-8所示。

代码清单10-8 DWRProxy与ComboBox整合

```
var info = Ext.data.Record.create([
    {name: 'id', type: 'int'},
    {name: 'name', type: 'string'}
]);

var store = new Ext.data.Store({
    proxy: new Ext.data.DWRProxy(infoManager.getItems, true),
    reader: new Ext.data.ListRangeReader({
        totalProperty: 'totalSize',
        root: 'data',
        id: 'id'
    }, info)
});

var combo = new Ext.form.ComboBox({
    store: store,
    displayField: 'name',
    valueField: 'id',
    triggerAction: 'all',
    typeAhead: true,
    mode: 'remote',
```

```
        emptyText: '请选择',
        selectOnFocus: true
    });
    combo.render('combo');
```

既可以用mode:'remote'和triggerAction:'all'在第一次选择时读取数据，也可以设置mode:'local'，然后手工操作store.load()并读取数据。

DWR要比Json-lib方便得多，而且DWR返回的数据可以直接作为JSON使用，使用Json-lib时还要面对无休无止的循环引用。

这个示例稍微复杂一些，因为包括依赖jar包、class、XML和JSP，所以示例单独放在10.store/dwr2/下，请将它们复制到tomcat的webapps下，然后再使用浏览器访问。

注意　EXT 3.0提供了新型的Ext Direct组件，专门用于解决不同平台下数据绑定与数据传输问题，它和DWR一样也是属于远程方法调用的范畴，但是Ext Direct提供了不同平台的服务端实现，因此可以运行在Java、.Net、PHP等主流平台之上，不同于DWR只支持Java平台。

10.11　localXHR 支持本地使用 Ajax

Ajax不能在本地文件系统中使用，必须把数据放到服务器上。无论是IIS、Apache、Tomcat，还是你熟悉的其他服务器，只要支持HTTP协议，就可以使用EXT中的Ajax。

至于本地为何不能用Ajax，主要是因为Ajax要判断HTTP响应返回的状态，只有status=200时才认为这次请求是成功的，因此必须先启动服务器才能对页面中的Ajax操作进行测试。

不过事情没有绝对的，下面我们就来介绍一个无需启动服务器就允许Ajax支持访问读取文件内容的插件localXHR。localXHR所作的就是强行修改响应状态，让Ajax的请求响应在没有启动服务器的情况下也可以继续进行。

下面我们来分析一下localXHR的源代码。

- 加入了一个forceActiveX属性，默认是false，它用来控制是否强制使用activex，activex是在IE下专用的。
- 修改了createXhrObject函数，只是在最开始处加了一条判断语句，如下所示：

```
if(Ext.isIE7 && !!this.forceActiveX){throw("IE7forceActiveX");}
```

- 增加了getHttpStatus函数，这是为了处理HTTP的响应状态，如代码清单10-9所示。

代码清单10-9　处理HTTP响应状态

```
getHttpStatus: function(reqObj){
    var statObj = {
        status:0
        ,statusText:''
        ,isError:false
        ,isLocal:false
        ,isOK:false
```

```
        };
        try {
            if(!reqObj)throw('noobj');
            statObj.status = reqObj.status || 0;

            statObj.isLocal = !reqObj.status && location.protocol == "file:" ||
                              Ext.isSafari && reqObj.status == undefined;

            statObj.statusText = reqObj.statusText || '';

            statObj.isOK = (statObj.isLocal ||
                           (statObj.status > 199 && statObj.status < 300) ||
                            statObj.status == 304);

        } catch(e){
            //status may not avail/valid yet.
            statObj.isError = true;
        }

        return statObj;
    },
```

代码清单10-9为状态增添了更多语义，status表示状态值，statusText表示状态描述，isError表示是否有错误，isLocal表示是否在本地进行Ajax访问，isOK表示操作是否成功。

判断isLocal是否为本地的有两种方法：reqObj没有status，而且请求协议是file:；浏览器是Safari，而且reqObj.status没有定义。

statObj中的isOK属性用来判断此次请求是否成功。判断请求是否成功的条件很多，例如：isLocal的属性为true、响应状态值在199到300之间、响应状态值是304等。如果处理过程中出现了异常，就会将isError属性设置为true，最后会把配置好的statObj对象返回，等待下一个步骤的处理。

localXHR.js对handleTransactionResponse函数进行了简化。因为增加的getHttpStatus函数很好地封装了与请求相关的各种状态信息，所以在handleTransactionResponse函数中我们不会看到让人头晕目眩的响应状态代码。取而代之的是isError和isOK这些更容易理解的属性，localXHR.js直接使用这些属性来处理响应。

createResponseObject函数被大大强化了。其实前半部分都是一样的，localXHR.js中对isLocal做了大量的处理，响应中的responseText可以从连接中获得。如果需要XML，它就使用ActiveXObject("Microsoft.XMLDOM")或new DOMParser()把responseText解析成XML放到response里，响应状态也是重新计算的，这样就能让Ajax正常调用了。

最后处理的是asyncRequest函数。如果在异步请求时出现异常，就调用handleTransactionResponse返回响应，然后根据各种情况稍微修改header属性。

我们来看看下面这行代码：

```
Ext.lib.Ajax.forceActiveX = (document.location.protocol == 'file:');
```

如果协议是file:，就强制使用activex。

注意 EXT-3.0重写了底层Ajax功能的实现，使用大量匿名函数替代了之前2.x版本中的单例形式，所以原本用于EXT-2.x中的localXHR.js已经无法用在EXT-3.0中了，因此我们基于EXT-3.0重新编写了localXHR，使它可以支持EXT-3.0中本地Ajax的请求。

10.12 小结

本章系统地讲述了Ext.data包中的各个类的功能和使用方式，还讨论了如何将EXT与DWR通过自定义的proxy相结合的示例。同时介绍了如何使用Ext.data.Connection与后台进行数据交互，还专门介绍了它的子类Ext.Ajax，并讨论了EXT中Ajax的应用以及在回调函数中使用scope或createDelegate()解决this的问题。

接着详细介绍了类Ext.data.Record和Ext.data.Store的功能和使用方法，这两个类结合起来形成了Ext.data中的主体数据模型，很多组件（包括表格和ComboBox）都是建立在它们之上的。除此之外，还讨论了常用的proxy、reader、store：SimpleStore和JsonStore，以及它们的应用场景。

最后介绍了扩展插件localXHR.js，它可以解决EXT中Ajax无法访问本地文件的问题。

第 11 章

实用工具

本章内容

- EXT提供的常用函数
- 使用DomHelper和Template动态生成HTML
- 使用Ext.Utils.CSS切换主题
- 悬停提示
- 使用Ext.state保存状态
- 使用fx实现动画效果
- 局部更新网页内容
- 使用Ext.util.Format对数据进行格式化
- 使用Ext.util.CSS管理CSS样式
- 使用Ext.util.ClickRepeater处理点击事件
- 使用Ext.util.DelayedTask延时执行函数
- 使用Ext.util.TaskRunner执行循环任务
- 混合型集合Ext.util.MixedCollection
- 使用Ext.util.TextMetrices获得文本所占的高度和宽度
- 使用Ext.KeyNav处理导航按键
- 使用Ext.KeyMap为对象绑定按键功能
- 扩展
- 门户组件Ext.ux.Portal
- 桌面组件Ext.Desktop

11.1 EXT 提供的常用函数

EXT提供了许多常用函数和工具，通过这些函数可以避免许多重复的工作。比如可以利用Ext.isGecko、Ext.isIE、Ext.isIE6、Ext.isIE7、Ext.isOpera、Ext.isSafari和Ext.isSafari2直接判断用户当前使用的浏览器类型。通过这些函数也可以实现：获取HTML中的元素、批量查询DOM元素、编码或解析JSON数据、实现EXT中类的继承与属性复制，以及更灵活地定义命名空间，等等。现在，我们来看一下如何使用这些常用的函数和工具。

11.1.1 onReady 函数

在使用EXT之前，我们都要面对页面加载的问题。因为EXT需要操作HTML中的DOM内容，只有页面完全下载到客户端并被浏览器完全解析后，我们才能启动EXT执行预先设置的功能。但是，我们如何才能得知页面在何时被解析完毕呢？

我们可以通过EXT提供的onReady函数来实现这项功能，利用它来注册HTML内容并加载完成后所需执行的代码。onReady函数有3个参数：第一个参数是必须的，表示HTML加载完成后需要执行的函数；第二个参数表示函数的作用域；第三个参数表示函数执行的一些其他操作，例如延迟时间（以毫秒为单位）等。大多数情况下，我们只需要用到第一个参数。

在下面的示例中，代码会在页面加载后执行init()函数，然后弹出提示信息"Everything is ready."，如下面的代码所示：

```
function init(){
    alert("Everything is ready.");
}
Ext.onReady(init);
```

我们可以在一个页面中调用onReady注册多个处理函数，这些函数会被放到事件队列中，在HTML加载完成后依次执行。onReady函数的第三个参数是与处理函数执行相关的一些特殊属性的对象，其中包含delay、single、buffer等属性。例如，在上面的代码中添加下面这行代码：

```
Ext.onReady(function(){alert("2")},this,{delay:5000});
```

页面加载完成后，会先执行init函数中的代码，5秒后再执行上面代码中onReady注册的匿名函数。

onReady函数的第二个参数被称作函数作用域，作用域是JavaScript中的一个比较关键的特性。下面的代码演示了如何在onReady函数中使用作用域：

```
var target1={value:"this is target1"};
var target2={value:"this is target2"};
function init() {
        alert(this.value);
}
Ext.onReady(init,target1);
Ext.onReady(init,target2);
```

上面的代码直接调用EXT类的onReady方法，并指定在页面加载完毕后执行init()函数。init()函数的作用是输出当前对象的value属性值，这里的this相当于对作用域的引用。执行这段代码后便可以看到onReady中作用域的效果。第一次用target1作为init()函数的作用域，所以弹出框中显示的this.value值是"this is target1"。第二次用target2作为init()函数的作用域，所以弹出框中显示的this.value值是"this is target2"。

11.1.2 get 函数

EXT中包含了几个以get开头的函数，这些函数可以用来获取HTML中的DOM对象、当前

HTML中的组件和EXT元素等。但是，在使用时要注意区分获取的对象的类型。

1. get函数

get函数用来获取一个EXT元素，也就是类型为Ext.Element的对象。Ext.Element类是EXT对DOM的封装，每个Element对象都对应着HTML中的一个DOM元素。我们可以为每一个DOM创建一个对应的Element对象，并通过Element对象中的函数来实现对DOM的指定操作。例如，可以使用hide函数隐藏元素，使用initDD函数为指定的DOM添加拖放特性等。get函数其实是Ext.Element.get的简写形式。

get函数中只有一个参数，但这个参数可以表示多种含义。它可以是DOM节点的id，也可以是一个Element，或者是一个DOM节点对象。我们来看看下面的示例代码：

```
Ext.onReady(function(){
  var e=new Ext.Element("hello");
  alert(Ext.get("hello"));
  alert(Ext.get(document.getElementById("hello")));
  alert(Ext.get(e));
});
```

对应的HTML页面中包含一个id为hello的div，代码如下所示：

```
<div id="hello">hello world</div>
```

从上面的示例中可以看出，Ext.get("hello")、Ext.get(document.getElementById("hello"))、Ext.get(e)这3个函数都可以获取一个与DOM节点hello对应的EXT元素。get函数可以自动判断参数的类型，最终返回我们需要的结果。

Element与document.getElementById("myDiv")获取的对象是不一样的。虽然可以使用以前的方式获得指定id的元素，但是会失去EXT提供的各种常用操作，如动画、定位、CSS、事件和拖放等。如果你想通过EXT的get函数获得指定id在HTML中对应的实际DOM对象，只需要使用Ext.get(id).dom。

下面我们来看一个稍微复杂点的示例，具体步骤如下所示。

(1) 先获得一个Element。

```
var myDiv = Ext.get('myDiv');
```

这里传入参数的是一个id，我们需要先在HTML中添加代码<div id="myDiv"></div>，然后用Ext.get('myDiv')获取这个div对应的Element对象。现在这个对象已经存放到EXT的缓存中了，当需要再次使用它时，可以直接从缓存中提取这个对象，不需要重复执行封装操作，这在很大程度上提高了程序的运行效率。

(2) 用获得的Element对象定义一个简单的动画效果，如下面的代码所示：

```
myDiv.hightlight();         //突出显示，然后渐退
myDiv.addClass('red');      //指定样式表
myDiv.center();             //居中显示
myDiv.setOpacity(.25);      //半透明效果
```

(3) 实现渐变动画效果，代码如下所示：

```
myDiv.setWidth(100);
```

虽然使用setWidth函数可以直接设置myDiv的宽度，但是无法实现渐变的动画效果。

```
myDiv.setWidth(100, true);
```

这里只需要为setWidth函数增加第二个参数，就相当于打开了动画的开关。现在我们可以看到myDiv的宽度从初始大小逐渐变成了100像素，如此简单就实现了动画效果。

还可以控制动画的效果，如下面的代码所示：

```
myDiv.setWidth(100, {
    duration: 2,
    callback: function(){alert('渐变完成')},
    scope: this
});
```

duration表示间隔，数字越大移动越慢，callback是动画完成后执行的回调函数，scope是callback的作用域。这些选项都会影响动画的最终效果。

示例在11.util/01-02-01.html中。

2. getCmp函数

getCmp函数用来获得一个EXT组件，也就是一个已经在页面中被初始化了的Component或其子类的对象，getCmp函数可以根据指定的id获得对应的Ext.Component。实际上，Ext.getCmp()是Ext.ComponentMgr.get()的简写形式。EXT中创建的每个组件都会注册到ComponentMgr中，只要得到它的id，就可以获得对应的组件，如下面的代码所示。

```
Ext.onReady(function(){
var h=new Ext.Panel({
      id:"h2",
      title:"旧的标题",
      renderTo:"hello",
      width:300,
      height:200}
);
Ext.getCmp("h2").setTitle("新的标题");
});
```

上面的代码使用Ext.getCmp("h2")来获得id为h2的组件，并调用其setTitle方法来设置该面板的标题。通过这种方法，我们可以获得之前已经创建的任意组件，并执行相应的操作。

3. getDom函数

getDom函数能够获得文档中的DOM节点，它只包含一个参数，该参数可以是DOM节点的id、DOM节点的对象或DOM节点对应的EXT元素等。Ext.getDom()可以看作Ext.get().dom的等同形式，如下面的代码所示：

```
Ext.onReady(function(){
    var e=new Ext.Element("hello");
    Ext.getDom("hello");
    Ext.getDom(e);
    Ext.getDom(e.dom);
});
```

对应的HTML代码如下所示：

```
<div id="hello">tttt</div>
```

在上面的代码中，Ext.getDom("hello")、Ext.getDom(e)、Ext.getDom(e.dom)3条语句返回的是同一个DOM节点对象。getDom函数将根据参数的类型自动获得我们所需的结果。

4. getBody函数

getBody函数可以直接获取文档中与document.body这个DOM节点对应的EXT元素。实质就是把document.body对象封装成EXT元素对象并作为结果返回，该方法不带任何参数。例如，下面的代码会把面板h直接渲染到文档的body元素中：

```
Ext.onReady(function()
var h=new Ext.Panel({title:"测试",width:300,height:200});
h.render(Ext.getBody());
});
```

我们可以把它当作快速更新页面显示组件的方法。

5. getDoc函数

getDoc函数可以将当前HTML文档对象（也就是document对象）封装成EXT的Element对象并返回，该方法不带任何参数。

与getBody函数类似，getDoc为我们提供了快速获取document对象的方法，只不过getBody函数获取的body部分只能用于显示，而document对象包含了更多对页面的操作。

11.1.3 query 函数和 select 函数

上一节讨论的函数都只能获得单个元素。在需要批量获得元素的情况下，就需要用到query和select函数了。函数query是Ext.DomQuery.select()的简写方式。

1. query函数和select函数的基本应用

query函数和select函数的区别：前者返回搜索到的元素数组，后者返回的是一个Ext.CompositeElement类型的对象。对于query返回的数组，我们只能循环迭代处理；而对于select返回的对象，我们还可以调用它提供的各种函数并执行特殊的操作。

首先，我们来了解一下query和select函数的基本操作方式。首先，创建一个包含<p>标签和<div>标签的HTML页面，让<p>标签和<div>标签交错显示，这样能更好地演示操作的效果。页面内容如下面的代码所示：

```
<p>p1</p>
<br>
<div class="red">div.red1</div>
<br>
<p>p2</p>
<br>
<div class="red">div.red2</div>
<br>
<p>p3</p>
<br>
```

我们希望获取HTML中所有的<p>元素，然后让这些<p>元素都突出显示。

使用select函数的情况如下：

```
Ext.select('p').highlight();
```

使用query函数的情况如下：

```
function selectP() {
    var array = Ext.query('p');
    for (var i = 0; i < array.length; i++) {
        Ext.get(array[i]).highlight();
    }
}
```

进一步，如果我们希望获得拥有class='red'样式的div元素，也可以通过query和select函数获得。

使用select函数的情况如下：

```
Ext.select('div.red').highlight();
```

使用query函数的情况如下：

```
function selectDiv() {
    var array = Ext.query('div.red');
    for (var i = 0; i < array.length; i++) {
        Ext.get(array[i]).highlight();
    }
}
```

由此可见，我们可以在select函数返回的Ext.CompositeElement对象上直接执行对应的功能函数，它会自动对包含的所有子元素产生作用。query函数返回的都是原生DOM对象，需要我们先手工遍历查询对象，然后再将它们转换成对应的Ext.Element对象，这样才能实现各种功能函数。

示例见11.util/01-03-01.html和11.util/01-03-02.html。

2. CSS元素选择符

在query和select函数中，我们通过CSS元素选择符来批量获取HTML页面中的元素，如下面的示例所示。

- Ext.query('span');：获得HTML中所有的span标签。
- Ext.query('span', 'foo');：获得id="foo"元素下的所有span标签。
- Ext.query('#foo');：获得id="foo"的标签。
- Ext.query('.foo');：获得所有class="foo"的标签。
- Ext.query('*');：使用了通配符，它会获得HTML中的所有标签。
- Ext.query('div p');：获得处于div标签下的所有p标签。
- Ext.query(div,p');：获得所有的div和p标签。

通过上面列举的这些示例，我们对CSS元素选择符有了一些初步的了解。

CSS元素选择符中有3种最基本的查找方式：第一种方式是直接输入span和div等标签的名称，它会去HTML中查找与其名称相同的所有标签；第二种方式是使用"#foo"，这种以#符号开头的形式表示我们查找的是一个id="foo"的元素，它会查找对应id的元素，将这个元素作为结果返回；第三种方式是使用".foo"，请注意开头的句点，这种情况下它会去查找所有

class="foo"的元素，并将它们作为结果返回。

除了上面3种基本的查询标签的方式，CSS元素选择符还支持使用通配符。在上面的示例中，我们就使用了星号（*）来匹配所有的标签。我们还可以使用逗号（,）一次指定多个标签，比如Ext.query（"div,p"）就会同时查找div和p两种标签，这样我们就可以一次获得多种标签组合的结果了。

CSS元素选择符还支持按层次进行查找。例如上例中的Ext.query("div span")，它就会获得所有在div标签下的span标签，这样我们就可以忽略掉处于其他结构下的span标签。如果有一些span标签没有包含在div标签内，就不会被放到返回的结果里。我们也可以指定一个查询的根结点，比如query('span', 'foo');就是查找所有包含在id="foo"节点下的span标签。默认情况下，我们会从body标签开始查找，通过指定根结点的位置，还可以缩小范围，从而获得更加精确的查询结果。

示例在11.util/01-03-03.html中。

3. CSS属性选择符

除了可以根据HTML中的标签、id、class进行查找以外，我们还可以根据HTML中元素的属性进行查找，如下面的示例所示。

- Ext.query('*[name]');：获得所有包含name属性的元素。
- Ext.query('*[name=bar]');：获得所有name属性等于bar的元素。
- Ext.query('*[name!=bar]');：获得所有name属性不等于bar的元素。
- Ext.query('*[name^=b]');：获得所有name属性以b开头的元素。
- Ext.query('*[name$=r]');：获得所有name属性以r结尾的元素。
- Ext.query('*[name*=a]');：获得所有name属性中含有a的元素。

CSS属性选择符是包含在中括号[]中的表达式，上例中所有的[]前面都使用了“*”通配符，表示我们将遍历所有标签来获得包含指定属性的元素。

我们可以直接使用属性名称来获得拥有这一属性的元素，也可以通过表达式来约束获得对应属性值的元素。在上面的示例中，表达式中使用了=、!=、^=、$=、*=等操作符，通过这些表达式我们就可以获得需要的元素了。

示例见11.util/01-03-04.html。

4. CSS值选择符

我们还可以对HTML元素中style属性包含的CSS样式的数值进行查找，如下面的示例所示。

- Ext.query('*{color=red}');：获得所有style="color:red"的标签。
- Ext.query('*{color=red} *{color=pink}');：获得所有style="color:red"元素下style="color:pink"的标签。
- Ext.query('*{color!=red}');：获得所有color的值不等于red的标签。
- Ext.query('*{color^=yel}');：获得所有color的值以r开头的标签。
- Ext.query('*{color$=ow}');：获得所有color的值以d结尾的标签。
- Ext.query('*{color*=ow}');：获得所有color的值包含a的标签。

CSS值选择符是包含在大括号{}中的表达式，上例中所有的{}前面都使用了“*”通配符，

表示我们将遍历所有标签来获得包含指定style值的元素。需要注意的是，我们这次使用的style值表达式与实际中的HTML标签的属性有些差异。例如，Ext.query('*{color=red}')对应到HTML标签中，获得的结果应该是包含了style="color:red"的各种标签，这里的CSS样式中的color的值就是我们所需要查找的部分。

我们可以通过表达式来约束获得对应style值的元素。在上面的示例中，表达式中也使用了=、!=、^=、$=、*=等操作符，通过这些表达式就可以获得所需的元素了。

示例见11.util/01-03-05.html。

11.1.4 encode 函数和 decode 函数

encode和decode函数是专门用来对JSON数据进行编码和解码的函数。Ext.encode()对应的解码方法为Ext.decode()。

encode函数的作用是对对象、数组或其他值进行编码，将对象转换成JSON字符串的形式，如下面的代码所示：

```
Ext.encode(obj) == '{"prop1":"a0~`!@#$%^&*()-_+={}[]|\\:;\"\',.?/","prop2":
["x","y"],"prop3":{"nestedProp1":"abc","nestedProp2":456}}'
```

上例中，我们用Ext.encode函数将一个简单的对象转换为JSON格式的字符串。这样做的主要目的是把JavaScript中的对象通过HTTP协议发送到后台服务器并进行相应处理。因为HTTP协议只能发送字符串形式的参数，所以无法将JavaScript中的对象直接传递给后台，这就需要先用Ext.encode函数对JavaScript对象进行编码，生成对应的JSON格式的字符串，再发送给后台服务器处理。

不过，如果希望将JSON转换为可通过HTTP发送的有效字符串形式，还需要再次对它进行编码。因为HTTP规范规定，HTTP请求只能发送ISO-8859-1编码的字符，所以像中文这种无法使用ISO-8859-1编码的字符，还需要先转换成ISO-8859-1编码格式才能通过HTTP协议传输，如下所示。

```
encodeURIComponent(Ext.encode(obj)) == "%7B%22prop1%22%3A%22a0~%60!%40%23%24%25%
5E%26*()-_%2B%3D%7B%7D%5B%5D%7C%5C%5C%3A%3B%5C%22'%2C.%3F%2F%22%2C%22prop2%22%3A%5
B%22x%22%2C%22y%22%5D%2C%22prop3%22%3A%7B%22nestedProp1%22%3A%22abc%22%2C%22nested
Prop2%22%3A456%7D%7D"
```

现在，我们可以把经过两次编码后获得的最终参数绑定到url上，然后传递给后台服务器进行处理。只需要像下面这样，直接把编码后的结果附加到url的后面。

```
"http://url.com?json=" + encodeURIComponent(Ext.encode(obj))
```

也可以使用POST方式的请求，先用编码后的参数结果生成对应的参数，然后将这些参数附加到请求的body部分，发送给后台服务器进行处理。

```
"json=" + encodeURIComponent(Ext.encode(obj))
```

除以上方法外，我们还可以用urlEncode()函数对参数进行处理，它会把JSON对象转换成HTTP要求的参数形式，并对参数值部分进行编码，最终得到的字符串就可以直接拼接到请求url上了，如下面代码所示：

```
Ext.urlEncode({ json: Ext.encode(obj)}) == "json=%7B%22prop1%22%3A%22a0~%60!%40%23%24%25%5E%26*()-_%2B%3D%7B%7D%5B%5D%7C%5C%5C%3A%3B%5C%22'%2C.%3F%2F%22%2C%22prop2%22%3A%5B%22x%22%2C%22y%22%5D%2C%22prop3%22%3A%7B%22nestedProp1%22%3A%22abc%22%2C%22nestedProp2%22%3A456%7D%7D"
```

无论使用上述哪种做法，在请求发送至后台服务器后，服务器对接收到的参数进行解析，最后会得到如下所示的参数形式：

```
{"prop1":"a0~`!@#$%^&*()-_+={}[]|\\:;\"',.?/","prop2":["x","y"],"prop3":
{"nestedProp1":"abc","nestedProp2":456}}
```

这样我们就可以直接调用JSON中保存的数据了。

Ext.decode()函数的作用是将字符串解析为JSON对象，但被解析的字符串的格式不能是任意的，它必须符合JSON要求的字符串格式。如果字符串的格式是无效的，该函数将抛出语法错误，实际应用过程如代码清单11-1所示。

代码清单11-1　使用decode()解析服务器返回的内容

```
Function successFn(response,options){
var obj= Ext.decode(response.responseText);
alert(obj.username);
}
function failureFn(response,options) {
alert('请求失败了');
}
Ext.Ajax.request({
  url: 'form.jsp',
  success: successFn,
  failure: failureFn,
   params: {dsname: 'INSERT' }
});
```

这里的回调函数返回的参数是一个XHR（即XmlHttpRequest）对象，我们可以通过该对象的responseText或responseXML属性获得由服务器端返回的数据信息。在Ajax应用中，我们经常会让服务器端返回JSON数据。由于JSON数据是字符串，因此在程序中需要先把它解析成JavaScript对象，然后才可以使用。要把JSON数据解析成JavaScript对象，可以直接使用Ext.decode函数。

假设后台服务器返回的JSON数据对象如下所示：

```
{
username: "wt",
times: 1
}
```

对象中包含两个属性：usename和times，username为"wt"，times为1。后台服务器会将这些数据转换成字符串发送给前台页面，前台页面在接收到这些数据后可以使用decode()函数进行如下处理，如下面的代码所示。

```
function successFn(response,options){
var obj= Ext.decode(response.responseText) ;
alert(obj.username);
}
```

我们先通过response.reponseText获得后台服务器端返回的响应数据，这个响应数据是符合JSON格式要求的字符串。只要解析这段JSON字符串，就可以得到JavaScript的对象，之后就可以像操作普通对象一样从它里面获得需要的数据。

我们先用Ext.decode()函数对响应数据进行解码，得到结果对象obj，然后从obj中获得username属性，最终得到的就是从服务器端传递过来的username的参数值"wt"。

11.1.5　extend 函数

在介绍Ext.extend()函数之前，我们先来看一下如何使用传统方式在JavaScript中实现类的继承操作。

首先定义一个父类，如下面代码所示：

```
var BaseClass = function() {
    // 未实现
};

BaseClass.prototype.someMethod = function() {
    // 未实现
};

BaseClass.prototype.overridenMethod = function() {
    // 未实现
}
```

在上面代码中，我们定义了一个名为BaseClass的类，然后为BaseClass定义了两个函数：someMethod()和overridenMethod()。然后定义一个BaseClass的子类，可以直接从BaseClass中继承所有的属性和函数，如下面的代码所示：

```
var SubClass = function() {
    BaseClass.call(this);
};

SubClass.prototype = new BaseClass();

SubClass.prototype.newMethod = function() {
    // 未实现
};

SubClass.prototype.overridenMethod = function() {
    // 未实现
};
```

在上面代码中，SubClass的构造函数首先调用BaseClass的构造函数初始化数据，然后通过SubClass.prototype = new BaseClass();这条语句让SubClass类获得BaseClass中的所有属性和函数，这样就实现了JavaScript中的继承操作。在此之后，我们就可以操作SubClass的prototype，为子类添加新函数或者覆写父类中的同名函数。

在EXT中使用Ext.extend()函数实现继承功能的方法如下面的代码所示：

```
var SubClass = function() {
```

```
SubClass.superclass.constructor.call(this);
};
Ext.extend(SubClass, BaseClass, {
newMethod : function() {},
overriddenMethod : function() {}
};
```

Ext.extend()函数提供了直接访问父类构造函数的途径，通过SubClass.superclass.constructor.call(this)就可以直接调用父类的构造函数，这个函数的第一个参数总是this，以确保父类的构造函数在子类的作用域里工作。

如果父类的构造函数需要传入参数，我们也可以将所需的参数直接传递给它，如下面的代码所示：

```
SubClass.superclass.constructor.call(this, config);
```

这样，我们就得到了一个继承了父类的所有属性和函数的子类。在EXT中，我们用extend函数可以轻易实现面向对象程序设计中的继承功能。

11.1.6 apply 函数和 applyIf 函数

Ext.apply函数的作用是将一个对象中的所有属性都复制到另一个对象中，如下面的代码所示：

```
var a= {desc:  '123'};
var b= { desc:  '456'};
Ext.apply(b, a);
alert(b.desc);
```

Ext.applyIf与Ext.apply的作用类似，它们的区别在于，如果某个属性在目标对象中已经存在，则Ext.applyIf不会将它覆盖，如下面的代码所示：

```
var a= {desc:  '123'};
var b= { desc:  '456'};
Ext.applyIf(b, a);
alert(b.desc);
```

最后显示的结果是：b.desc的值依然是"456"，这说明applyIf函数不会破坏对象中的初始数据，这在某些情况下是非常有用的。

11

11.1.7 namespace 函数

Ext.namespace函数的作用是把传入的参数转换成对象，使用该方法的主要目的是区分名称相同的类，这与Java中的package的作用类似。我们先看看下面的代码：

```
namespace: function(){
    var a = arguments, o = null, i, j, d, rt;
    for (i = 0; i < a.length; ++i) {
        d = a[i].split(".");
        rt = d[0];
        eval('if (typeof ' + rt + ' == "undefined"){' + rt + ' = {};} o = ' + rt + ';');
        for (j = 1; j < d.length; ++j) {
```

```
                o[d[j]] = o[d[j]] ||{};
                o = o[d[j]];
            }
        }
    }
```

从上面的代码中可以看出，如果传入的字符串参数是以“.”分隔的，EXT就会根据定义的结构创建多个对象，如下所示：

```
Ext.namespace('system.corp');
```

在上面的代码中，我们共创建了两个对象，分别是system对象和system.corp对象。其中corp对象是作为system对象的一个属性被创建的，这相当于执行了下面的代码：

```
system = …{};
system.corp = …{};
```

通过namespace函数，我们在定义自己的类时就可以像使用Java中的包一样直接引用一长串类名了。这样可以保证每个人定义的类都放在不同的命名空间中，避免了命名冲突。

```
Ext.namespace('system.corp');
system.corp.ManageCorp = function() …{
  //未实现
}
```

如果还想定义一个名为ManageCorp的类，可以使用不同的namespace来进行区分，这样两个类就不会冲突了，如下面的代码所示：

```
Ext.namespace('system.admin');
system.admin.ManageCorp = function() …{
  //未实现
}
```

还可以直接使用namespace函数的简写形式ns()，如下所示：

```
Ext.ns('system.admin');
```

还可以同时为namespace函数传递多个参数，这样就可以同时创建多个命名空间，供以后的操作使用，如下所示：

```
Ext.ns('system.admin', 'system.corp');
```

11.1.8 Ext.isEmpty 函数

isEmpty()函数可以判断传入的参数值是否为空，当参数值为null、undefined或空字符串时，isEmpty()函数会返回true，否则会返回false。下面我们对不同参数值的情况进行逐一分析。

- 当参数为null时，isEmpty()返回true，如下面的代码所示：

```
var param = null;
alert("param = null, isEmpty: " + Ext.isEmpty(param));
```

- 当参数为undefined时，isEmpty()返回true，如下面的代码所示：

```
delete param;
alert("param = undefined, isEmpty: " + Ext.isEmpty(param));
```

- 当参数为空字符串时，isEmpty()返回true，如下面的代码所示：

```
var param = '';
alert("param = '', isEmpty: " + Ext.isEmpty(param));
```

- 还可以使用allowBlank参数，让isEmpty()在空字符串时返回false，如下面的代码所示：

```
alert("param = '', allowBlank, isEmpty: " + Ext.isEmpty(param, true));
```

- isEmpty()无法判断一个数组是否为空，因此，无论数组是否为空，它都会返回false，如下面的代码所示：

```
param = [];
alert("param = [], isEmpty: " + Ext.isEmpty(param));
```

- isEmpty()无法判断一个对象是否为空，因此，无论对象是否为空，它都会返回false，如下面的代码所示：

```
param = {};
alert("param = {}, isEmpty: " + Ext.isEmpty(param));
```

上面的内容讨论了使用isEmpty()判断变量值是否为空的情况，实际开发中我们可以直接使用它判断使用的变量是否为空。

示例文件在11.util/_01_01_08.html中。

11.1.9 **Ext.each** 函数

当需要对数组中的每个元素进行同一种操作时，可以使用Ext.each()函数，它会迭代循环数组，将每个元素都传入预先定义的回调函数中进行处理。

可以使用Ext.each()函数对整数数组中的数据执行求和操作，如下面的代码所示：

```
var array = [1, 2, 3, 4];
var sum = 0;
Ext.each(array, function(item) {
    sum += item;
});
alert(sum);
```

Ext.each()函数会迭代循环array数组，将array数组中的元素依次传入我们定义的回调函数中处理。回调函数中的item参数代表的就是每次循环获得的array数组中的元素，我们将获得的参数依次累加到sum变量中。整个迭代过程中，回调函数会被调用4次，最终sum中得到的就是数组中所有元素的总和。

如果回调函数中包含this引用，就需要在使用Ext.each()函数进行迭代循环时指定回调函数执行的范围，如下面的代码所示：

```
App = {
    sum: 0,
```

```
        fn: function(item) {
            this.sum += item;
        }
    };
    Ext.each(array, App.fn, App);
    alert(App.sum);
```

上述代码中，我们的回调函数定义在App对象中，它需要操作App的sum属性执行累加操作。当调用Ext.each()函数时，就需要使用第三个参数将App.fn执行的范围设置为App，这样回调函数执行时才能正确获得this引用，将数组中的数据累加到App的sum属性中。

一般情况下，回调函数只需要获取每次迭代循环得到的元素值。不过有时也需要获得当前循环的索引值，或者也需要在每次迭代循环时访问整个数组中的数据，这时可以使用Ext.each()为回调函数预留的另外两个参数，如下面的代码所示：

```
sum = 0;
Ext.each(array, function(item, index, allArray) {
    sum += item * index + allArray.length;(得到的sum是啥)
});
alert(sum);
```

这次回调函数中使用了3个参数，item代表当前迭代循环得到的元素，index代表当前的循环索引值，allArray代表正在执行迭代循环的整体数组。这样每次循环时都将当前的元素值乘以循环索引值，再加上整体数组的长度，将计算的结果累加到sum变量中。

示例文件在11.util/_01_01_09.html中。

11.1.10 **Ext.DomQuery**

Ext.DomQuery支持CSS 3选择器的大部分功能，并且提供了一些自定义选择符，它还能支持基础的XPath。DomQuery的select()函数将接受我们定制的查询表达式，将HTML中所有满足条件的标签作为结果返回。

假设我们有如下HTML代码：

```
<div id="root1">
  <span class="span1">span 1</span>
  <span class="span2">span 2</span>
  <span class="span1">span 3</span>
  <span class="span2">span 4</span>
  <span class="span1">span 5</span>
</div>
<div id="root2">
  <span class="span0">span 0</span>
</div>
```

页面显示效果如图11-1所示。

可以用DomQuery获得页面上所有的span标签，如下面的代码所示：

```
array = Ext.DomQuery.select('span');
Ext.each(array, function(elem) {
    elem.style.backgroundColor = 'red';
});
```

将查询得到的每个标签的背景颜色都设置为红色，页面显示效果如图11-2所示。

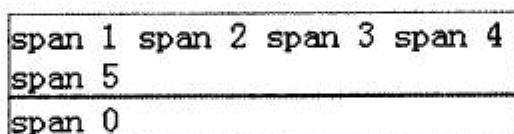


图11-1 示例HTML代码显示效果



图11-2 改变标签背景颜色后的显示效果

也可以获得所有位于id="root1"的div标签内部的span标签，如下面的代码所示：

```
array = Ext.DomQuery.select('div#root1>span');
Ext.each(array, function(elem) {
    elem.style.color = 'yellow';
});
```

还可以实现更复杂的功能，比如获得id="root1"的div下所有class为span1的span标签，如下面的代码所示：

```
array = Ext.DomQuery.select('div#root1>span.span1');
Ext.each(array, function(elem) {
    elem.style.backgroundColor = 'green';
});
```

在执行批量查询和批量更新操作时，DomQuery无疑是最佳选择。下面我们来看一下DomQuery支持的选择符和伪类，以及这些选择符和伪类的含义。

1. 元素选择符

- *：匹配所有元素。
- E：匹配标签名为E的元素，如<E></E>。
- E F：匹配所有包含在E标签中的F标签元素，如<E><F></F></E>中的<E>标签。
- E>F or E/F：匹配所有包含在E标签中，作为直接子节点的F标签元素，如E>F只能匹配<E><F></F></E>中的<E>标签而不能匹配<E><C><F></F></C></E>中的<E>标签。
- E+F：匹配所有直接前驱节点为F标签的E标签，如E+F只能匹配<E></E><F></F>中的<E>标签，而不能匹配<E></E><C></C><F></F>中的<E>标签。
- E~F：匹配所有前驱兄弟节点为F标签的E标签，如E~F既能匹配<E></E><F></F>中的<E>标签，也能匹配<E></E><C></C><F></F>中的<E>标签。

2. 属性选择符

@和单引号都是可选的，div[@foo='bar']和div[foo=bar]这两种写法都是有效的。

- E[foo]：匹配所有拥有foo属性的<E>标签，如<E foo=""></E>。
- E[foo=bar]：匹配所有foo属性值为bar的<E>标签，如<E foo="bar"></E>。
- E[foo^=bar]：匹配所有foo属性值以bar开头的<E>标签，如<E foo="barxx"></E>。
- E[foo$=bar]：匹配所有foo属性值以bar结尾的<E>标签，如<E foo="xxbar"></E>。
- E[foo*=bar]：匹配所有foo属性值中包含bar的<E>标签，如<E foo="xxbarxx"></E>。
- E[foo%=2]：匹配所有foo属性值可以被2整除的<E>标签，如<E foo="4"><E>。
- E[foo!=bar]：匹配所有foo属性值不为bar的<E>标签，如<E foo="xx"></E>。

11

3. 伪类

- E:first-child：匹配所有当前标签是本身父节点的第一个子节点的元素，对应<E><first></first><second></second><last></last></E>中的<fisrt>标签。
- E:last-child：匹配所有当前标签是本身父节点的最后一个子节点的元素，对应<E><first></first><second></second><last></last></E>中的<last>标签。
- E:nth-child(n)：匹配所有当前标签是本身父节点的第n个子节点的元素（索引从1开始计算），如E:nth-child(2)将获得第二个子节点，对应<E><first></first><second></second><last></last></E>中的<second>标签。
- E:nth-child(odd)：匹配所有当前标签是本身父节点的奇数子节点的元素，对应<E><first></first><second></second><last></last></E>中的<first></first>和<last></last>。
- E:nth-child(even)：匹配所有当前标签是本身父节点的偶数子节点的元素，对应<E><first></first><second></second><last></last></E>中的<second>标签。
- E:only-child：匹配所有当前标签是本身父节点的唯一子节点的元素，只对应<E><only></only></E>形式的<E>标签。
- E:checked：匹配所有当前标签的check属性为true的元素（比如单选框或复选框），如<E><input type="checkbox" checked></E>中的<E>标签。
- E:first：当前标签为结果集的第一个元素，如<E id="e1"></E><E id="e2"></E><E id="e3"></E>时，E:first返回的是id为e1的<E>标签。
- E:last：当前标签为结果集的最后一个元素，如<E id="e1"></E><E id="e2"></E><E id="e3"></E>时，E:last返回的是id为e3的<E>标签。
- E:nth(n)：当前标签为结果集的第n个元素（索引从1开始计算），如<E id="e1"></E><E id="e2"></E><E id="e3"></E>时，E:nth(2)返回的是id为e2的<E>标签。
- E:odd：当前标签为结果集中位于奇数位置的元素，如<E id="e1"></E><E id="e2"></E><E id="e3"></E>时，E:odd返回的是id为e1和e3的<E>标签。
- E:even：当前标签为结果集中位于偶数位置的元素，如<E id="e1"></E><E id="e2"></E><E id="e3"></E>时，E:even返回的是id为e2的<E>标签。
- E:contains(foo)：标签的innerHTML属性值包含"foo"，如<E>xxfooxx</E>。
- E:nodeValue(foo)：标签的textValue或nodeValue值为"foo"，如<E>foo</E>。
- E:not(S)：标签不匹配简单选择符S，如<E class="e1"></E><E class="e2"></E><E class="e3"></E>，E:not(E.e1)返回的是class为e2和e3的<E>标签。
- E:has(S)：标签有一个后继与简单选择符S匹配，如E:has(F.f)会匹配<E><F class="f"></F></E>中的<E>标签。
- E:next(S)：标签的后一个兄弟标签与简单选择符S匹配，如E:next(F.f)会匹配<E></E><F class="f"></F>中的<E>标签。
- E:prev(S)：标签的前一个兄弟标签与简单选择符S匹配，如E:prev(F.f)会匹配<F class="f"></F><E></E>中的<E>标签。

4. CSS值选择符

- E{display=none}：匹配所有CSS的"display"值为"none"的元素，如<Estyle ="display:none"></E>。
- E{display^=none}：匹配所有CSS的"display"值以"none"开头的元素，如<Estyle ="display:nonexx"></E>。
- E{display$=none}：匹配所有CSS的"display"值以"none"结尾的元素，如<Estyle ="display:xxnone"></E>。
- E{display*=none}：匹配所有CSS的"display"值包含"none"的元素，如<Estyle ="display:xxnonexx"></E>。
- E{display%=2}：匹配所有CSS的"display"值能被2整除的元素，如<Estyle ="display:4"></E>。
- E{display!=none}：匹配所有CSS的"display"值不为"none"的元素，如<Estyle ="display:xx"></E>。

了解了DomQuery所支持的上述CSS选择符和伪类之后，我们可以调用它提供的功能函数处理对应的问题。DomQuery主要提供了以下7个功能函数。

- compile()函数的作用是将CSS选择符编译成一个回调函数，用户可以为回调函数传递一个root参数，回调函数就会以root参数对应的元素为起点进行查询。
- filter()函数会根据指定的CSS选择符把数组中不匹配的元素过滤掉。
- is()函数会使用CSS选择符与传入的元素进行匹配，如果匹配成功就返回true，否则返回false。
- select()会根据CSS选择符和root根节点返回匹配的元素数组。
- selectNode()、selectNumber()、selectValue()获得的都是单一结果，其中selectNode()函数返回节点对象，selectValue()函数返回节点的值，selectNumber()函数将节点的值转换成数字再返回给用户。

11.2 使用 DomHelper 和 Template 动态生成 HTML

使用DOM的原生API动态生成HTML一直是一件令人头疼的事情。在EXT中，几乎所有组件都是动态生成的，现在我们就来看看EXT是如何动态生成HTML的。

11.2.1 使用 DomHelper 生成小片段

DomHelper可以帮助我们快速生成小块的HTML内容，很多情况下都可以直接使用它来实现动态生成HTML内容的功能。先来看看下面这段代码：

```
var list = Ext.DomHelper.append('parent', {tag: 'div', cls: 'red'});
```

这段代码表示向id="parent"的元素中添加一个div元素，并且为这个div指定名为“red”的CSS样式。

根据文档里的描述，DomHelper.append()函数的第二个参数都会成为新生成的元素的属

性，只有4个特殊的属性除外，如下所述。

- tag：指定生成的标签类型，如div和span等。
- cls：表示HTML标签中的class属性，因为class是JavaScript的关键字，所以不能直接使用。实际上，当用DOM为元素设置class属性时，应该使用classname。为了方便，在EXT中简写成了cls[①]。
- children：指定子节点，它的值是一个数组，数组中包含了对相应子节点的定义。
- html：对应标签的innerHTML值，如果觉得用children描述太烦琐，也可以直接使用html设置标签内包含的HTML内容。

除了append之外，DomHelper还有其他几个函数，通过它们可以指定将新节点添加到DOM中的各个位置。

为了比较效果，我们先制作一个初始页面，如图11-3所示。

原始的HTML是一个div，它下面有4个节点，其中第三个子节点还有自己的子节点，如下面的代码所示：



图11-3　测试DomHelper的初始页面

```
<div id="parent" style="border: 1px solid black;padding: 5px;margin: 5px;background:
lightgray;">
  <p id="child1">child1</p>
  <p id="child2">child2</p>
  <div id="child3">
    <p id="child5">inner child</p>
  </div>
  <p id="child3">child4</p>
</div>
```

(1) DomHelper.append()函数会将新生成的节点放到指定节点的最后面。

下面的代码将在id="parent"的节点的最后面添加一个<p>标签，并设置了新生成的标签的CSS样式和内部显示的HTML内容。

```
Ext.DomHelper.append('parent', {tag: 'p', cls: 'red', html: 'append child'});
```

上述代码的效果如图11-4所示。

(2) DomHelper.insertBefore()函数会将新生成的节点插入到指定节点前面。

下面的代码将在id="parent"的节点前面添加一个<p>标签，并设置了新生成的标签的CSS样式和内部显示的HTML内容。

```
Ext.DomHelper.insertBefore('parent',{tag:'p',cls:'red',html:'insert before child'})
```

上述代码的效果如图11-5所示。

① 在EXT中，类似这样的简写很多，例如DataStore简写为ds，DomHelper简写为dh，Element简写为el，ColumnModel简写为cm，SelectionModel简写为sm。

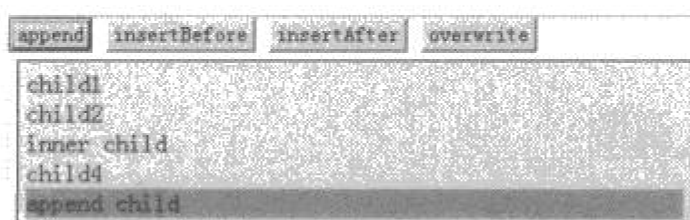


图11-4　append

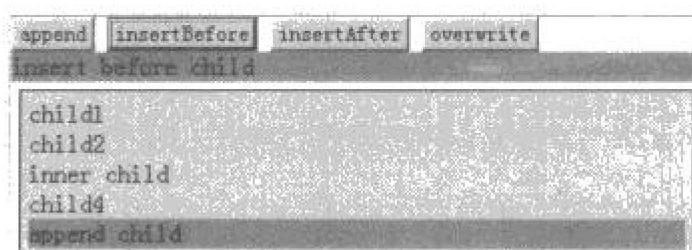


图11-5　insertBefore

(3) DomHelper.insertAfter()函数会将新生成的节点插入到指定节点后面。

下面的代码将在id="parent"的节点后面添加一个<p>标签，并设置新生成标签的CSS样式和内部显示的HTML内容。

```
Ext.DomHelper.insertAfter('parent',{tag:'p', cls: 'red', html: 'insert after child'})
```

上述代码的效果如图11-6所示。

(4) DomHelper.overwrite()函数会替换指定节点的innerHTML内容。

下面的代码将把id="child3"的节点的内容替换成一个<p>标签，并设置了新生成的标签的CSS样式和内部显示的HTML内容。

```
Ext.DomHelper.overwrite('child3', {tag: 'p', cls: 'red', html: 'overwrite child'})
```

上述代码的效果如图11-7所示。

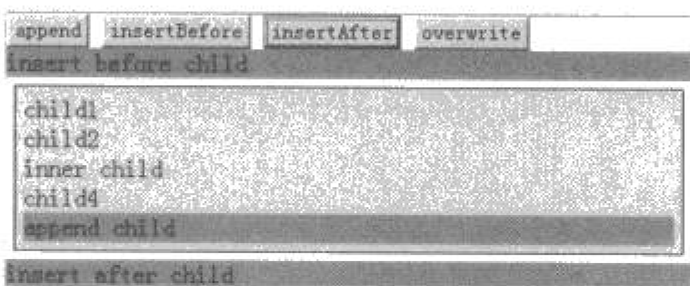


图11-6　insertAfter

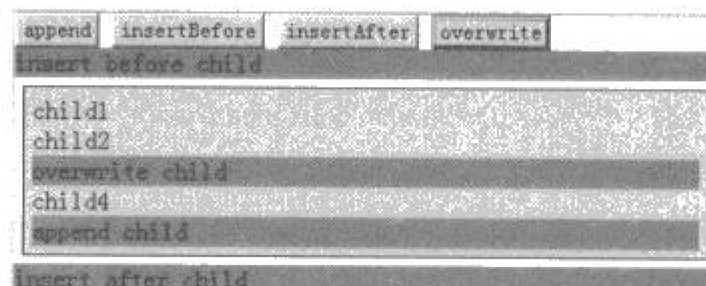


图11-7　overwrite

现在我们来讨论一下属性children的用法，如下面的代码所示：

```
Ext.DomHelper.append('parent', {
    tag: 'ul',
    style: 'background: white;list-style-type: disc;padding: 20px;',
    children: [
        {tag: 'li', html: 'li1'},
        {tag: 'li', html: 'li2'},
        {tag: 'li', html: 'li3'}
    ]
});
```

上述代码的结果是在id="parent"的节点中添加一个ul标签，这个ul标签中包含3个li标签，如图11-8所示。使用children属性，我们就可以直接生成一些复杂的DOM结构。

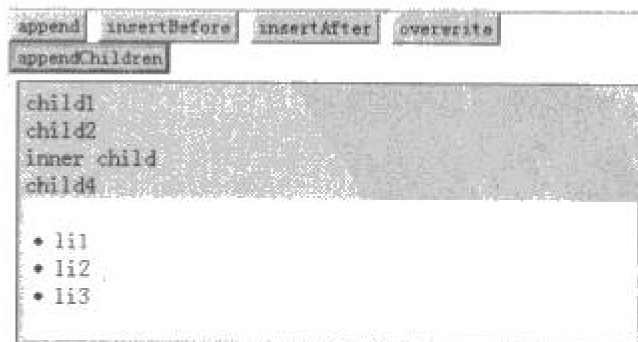


图11-8　insertChildren

示例在11.util/02-01-01.html中。

11.2.2 Ext.DomHelper.applyStyles 函数

Ext.DomHelper类的applyStyles()函数可以为指定的DOM元素设置指定的CSS样式。我们可以使用字符串、JSON对象或回调函数为对应的DOM元素设置CSS样式，如图11-9所示。



图11-9 使用applyStyles设置指定DOM元素的CSS样式

上例分别为3个DOM元素设置了背景颜色。

对于只需要设置单一样式的情况，可以直接使用字符串格式的参数"background-color:red"，如下面的代码所示：

```
Ext.DomHelper.applyStyles("child1", "background-color:red;");
```

对于设置多种样式的情况，可以使用JSON对象作为参数来设置CSS样式，如下面的代码所示：

```
Ext.DomHelper.applyStyles("child2", {
    'background-color':'green',
    'font-weight':'bold',
    'font-style':'italic',
    'text-decoration':'line-through'
});
```

对于需要经过某种逻辑计算才能决定设置何种CSS样式的情况，可以使用一个回调函数作为参数，不过这个回调函数返回的值必须符合上述的字符串或JSON对象类型的参数格式，如下面的代码所示：

```
Ext.DomHelper.applyStyles("child3", function() {
    var time = new Date().getTime();
    if (time % 2 == 0) {
        return "background-color:yellow";
    } else {
        return "background-color:blue";
    }
});
```

示例在11.utils/_01_02-01-02.html中。

11.2.3 Template（模板）

在一般情况下，用DomHelper就已经足以。但是在某些复杂的情况下，我们还是需要用模板来批量生成所需要的HTML。假设有这样一种情况：我们已经获得了一个包含3位男性和2位女性的信息的JSON对象，现在需要将这一组数据生成为HTML标签并显示到页面中，如下面的代码所示：

```
var data = [
    ['1','male','name1','descn1'],
    ['2','female','name2','descn2'],
```

```
        ['3','male','name3','descn3'],
        ['4','female','name4','descn4'],
        ['5','male','name5','descn5']
];
```

首先定义一个模板，如下面的代码所示：

```
var t = new Ext.Template(
    '<tr>',
        '<td>{0}</td>',
        '<td>{1}</td>',
        '<td>{2}</td>',
        '<td>{3}</td>',
    '</tr>'
);
t.compile();
```

注意，模板中的数据索引是从0开始的，每一行都包含4个元素，分别对应JSON中每一行的4个数据。创建新模板后，必须调用compile()函数进行编译，现在可以通过这个模板生成我们需要的HTML内容了，如下面的代码所示：

```
for (var i = 0; i < data.length; i++) {
    t.append('some-element', data[i]);
}
```

在append()函数中，第一个参数"some-element"对应着HTML中的一个表格，如下面的代码所示：

```
<table border="1">
    <tbody id="some-element">
        <tr>
            <td>id</td>
            <td>sex</td>
            <td>name</td>
            <td>descn</td>
        </tr>
    </tbody>
</table>
```

这样，我们使用EXT提供的模板将JSON数据的每一行都转换为HTML，并添加到指定的Table标签中，最终的显示结果就是包含5行数据的表格，如图11-10所示。

id	sex	name	descn
1	male	name1	descn1
2	female	name2	descn2
3	male	name3	descn3
4	female	name4	descn4
5	male	name5	descn5

图11-10 template生成的结果

除了上面演示的情况外，我们还可以在定义模板时使用Ext.util.Format中预定义的格式化函数对原始数据进行格式化。最常用的是trim和ellipsis，trim会去掉字符串的首尾空白，而ellipsis会在字符长度超过指定长度时自动截断字符串，并在末尾添加省略号。

在模板里使用这些函数也很简单，只需在指定数据的索引后使用冒号和函数名称即可。下面的示例演示了如何对第2列和第3列数据进行trim操作，并限制第4列数据最后只显示10个字符，如下面的代码所示：

```
var t = new Ext.Template(
    '<tr>',
        '<td>{0}</td>',
        '<td>{1:trim}</td>',
        '<td>{2:trim}</td>',
        '<td>{3:ellipsis(10)}</td>',
    '</tr>'
);
t.compile();
```

当然，即使EXT在设计时考虑得再周到，也不可能兼顾现实开发中可能出现的所有情况。幸运的是，EXT准备了一个自定义函数的接口，我们可以通过这个接口实现所需的功能。例如，根据性别的值在页面上显示对应的图片，实现过程如代码清单11-2所示。

代码清单11-2 使用模板的自定义接口

```
var t = new Ext.Template(
    '<tr>',
        '<td>{0}</td>',
        '<td>{1:this.renderSex}</td>',
        '<td>{2:trim}</td>',
        '<td>{3:ellipsis(10)}</td>',
    '</tr>'
);
t.renderSex = function(value) {
    if (value == 'male') {
        return "<span style='color:red;font-weight:bold;'>红男</span><img src='user_
        male.png' />";
    } else {
        return "<span style='color:green;font-weight:bold;'>绿女</span><img src='
        user_female.png' />";
    }
};
t.compile();
```

首先，我们在定义模板时使用{1:this.renderSex}的方式指定：处理第2列数据时必须调用我们自定义的renderSex函数。然后，为创建好的模板添加senderSex函数，这个函数的参数就是生成模板时对应的原始数据。对原始数据进行处理后，返回的数值将会显示在页面中。

通过自定义接口，我们实现了根据性别显示对应图片的功能，如图11-11所示。

id	sex	name	descn
1	红男	name1	descn1
2	绿女	name2	descn2
3	红男	name3	descn3
4	绿女	name4	descn4
5	红男	name5	descn5

图11-11 使用Template自定义接口的效果

对应的完整代码如下所示：

```
var data = [
    ['1','male','name1','descn1'],
    ['2','female','name2','descn2'],
    ['3','male','name3','descn3'],
    ['4','female','name4','descn4'],
    ['5','male','name5','descn5']
```

```
    ];

var t = new Ext.Template(
    '<tr>',
        '<td>{0}</td>',
        '<td>{1:this.renderSex}</td>',
        '<td>{2:trim}</td>',
        '<td>{3:ellipsis(10)}</td>',
    '</tr>'
);
t.renderSex = function(value) {
    if (value == 'male') {
        return "<span style='color:red;font-weight:bold;'>红男</span><img
        src='user_male.png' />";
        } else {
            return "<span style='color:green;font-weight:bold;'>绿女</span><img
            src='user_female.png' />";
    }
};
t.compile();

for (var i = 0; i < data.length; i++) {
    t.append(Ext.get('some-element'), data[i]);
}
```

示例在11.util/02-02-01.html中。

11.2.4 Ext.DomHelper.createTemplate 函数

createTemplate()函数可以直接创建一个模板对象，与DomHelper所支持的DOM对象相比，模板显得更加灵活。下面我们来看看如何使用createTemplate()函数将DomHelper和Template这两种HTML生成方式结合起来，如下面的代码所示：

```
var tpl = Ext.DomHelper.createTemplate(
    {tag: "tr", children: [
        {tag: 'td', html: '{0}'},
        {tag: 'td', html: '{1}'},
        {tag: 'td', html: '{2}'},
        {tag: 'td', html: '{3}'}
    ]}
);
tpl.compile();

var data = [
    ['1','male','name1','descn1'],
    ['2','female','name2','descn2'],
    ['3','male','name3','descn3'],
    ['4','female','name4','descn4'],
    ['5','male','name5','descn5']
];

for (var i = 0; i < data.length; i++) {
    tpl.append('some-element', data[i]);
}
```

createTemplate()方法接收JSON类型的对象，在内部将对象转换成HTML片段，随后使用HTML片段生成模板对象，这样返回的tpl变量经过编译就可以在后面的代码中直接使用了。用一个数组作为数据源，使用模板将生成的HTML结果插入id="some-element"的DOM元素中，最终的结果如图11-12所示。

11.2.5　复杂模板 XTemplate

XTemplate对Template的功能进行了增强，XTemplate不仅支持在模板内部使用子模板，还具有for和if的功能，可以让我们在模板中实现循环和判断等功能。

id	sex	name	descn
1	male	name1	descn1
2	female	name2	descn2
3	male	name3	descn3
4	female	name4	descn4
5	male	name5	descn5

图11-12　使用DomHelper.createTemplate()函数生成的HTML片段的结果

先看看如何用XTemplate输出一个下拉框，代码如下所示。

```
var data = {
    name: 's',
    size: 5,
    options: [
        {value: '河北', text: '河北省'},
        {value: '唐山', text: '唐山市'},
        {value: '路北', text: '路北区'}
    ]
};
```

data中主要包含两部分内容：主体部分的name和size，以及options数组。我们希望复制name和size两个属性并赋予<select>标签，然后用options数组中的数据生成<select>标签下的<option>。使用的XTemplate如下面的代码所示：

```
var t = new Ext.XTemplate(
    '<select name="{name}" size="{size}">',
        '<tpl for="options">
<option value="{value:trim}">{text:ellipsis(10)}</option>
</tpl>',
    '</select>'
);

t.append('f', data);
```

从上面的代码中可以看出，我们直接用{name}和{size}的方式在模板中指定了主体部分的name和size。随后，用<tpl>标签定义了一个子模板，for="options"属性表示我们要对原始数据中的options属性执行循环操作。这样，子模板就会自动调用options的数据，循环输出为<option>标签。

上面演示的只是XTemplate的一个普通功能，现在我们来看看它更强大的功能。

1. 操作简单数组

```
var data = {
    name: 's',
    size: 5,
    options: ['河北省', '唐山市', '路北区']
};
```

```
var t = new Ext.XTemplate(
    '<select name="{name}" size="{size}">',
        '<tpl for="options">',
            '<option value="{.:trim}">{.:ellipsis(10)}</option>',
        '</tpl>',
    '</select>'
);
```

我们将data.options中的数据改为包含3个字符串的简单数组，子模板tpl里直接使用{.}表示数组中的每一项，默认的格式化函数依然有效。

2. 默认选中奇数行

```
var t = new Ext.XTemplate(
  '<select name="{name}" size="{size}" multiple>',
        '<tpl for="options">',
            '<tpl if="xindex % 2 == 0">',
                '<option value="{.:trim}">{.:ellipsis(10)}</option>',
            '</tpl>',
            '<tpl if="xindex % 2 == 1">',
                '<option value="{.:trim}" selected>{.:ellipsis(10)}</option>',
            '</tpl>',
        '</tpl>',
    '</select>'
);
```

因为XTemplate中没有提供else功能，我们只好使用两个if来进行条件判断。这里使用的xindex是for内置的一个变量，它表示当前循环项的索引值。注意，xindex是从1开始计数的。

3. 简化判断

```
var t = new Ext.XTemplate(
    '<select name="{name}" size="{size}" multiple>',
        '<tpl for="options">',
            '<option value="{.:trim}"{[xindex % 2 == 1 ? " selected" : ""]}>{.:
            ellipsis(10)}</option>',
        '</tpl>',
    '</select>'
);
```

我们直接用一个三元判断替换了上一节中冗长的if判断，因为这个xindex并不是外部提供的数据，所以要在{}中再加上一个[]，在XTemplate中调用内部变量都要用这种方式。

表11-1列出了XTemplate中的内部变量以及它们的含义说明。

表11-1 XTemplate中的内部变量

变 量 名	说 明
values	当前范围内的变量，每次使用时tpl都会进入一个子模板，只能使用子模板对应的变量
parent	如果进入了子模板，就需要通过parent引用上一级模板里的变量
xindex	循环的索引值，从1开始
xcount	循环的总长度
fm	Ext.util.Format，想执行格式化操作，就直接调用它

示例在11.util/02-03-02.html中。

11.3 用 Ext.Utils.CSS 切换主题

相信大家对EXT精美的主题风格有非常深刻的印象，但是，再精美的主题风格，看多了也会审美疲劳。所以，EXT为我们提供了default和gray两种主题风格。如何才能在页面中动态切换这两种主题风格呢？

因为在EXT中是使用CSS样式表来定义页面的显示风格的，所以只要能动态修改页面中引用的CSS就可以实现主题切换。现在，我们可以用Ext.util.CSS.swapStyleSheet()函数来实现CSS切换，这个函数会动态修改指定id的CSS标签，指向不同的外部CSS文件，从而修改整个项目的样式，这样就达到了切换主题风格的目的，具体的操作步骤如下所示。

(1) 制作一个选择主题select，如下面的代码所示：

```
<select id="themeSelect">
    <option value="default">default</option>
    <option value="gray">gray</option>
</select>
```

(2) 为这个select添加监听change事件的函数，如下面的代码所示：

```
Ext.get("themeSelect").on("change", function(e) {
    var v = e.target.value;
    if (v == 'default') {
        Ext.util.CSS.swapStyleSheet("theme", "../../resources/css/ext-all.css");
    } else {
        Ext.util.CSS.swapStyleSheet("theme", "../../resources/css/xtheme-" +
        e.target.value + ".css");
    }
});
```

我们使用了函数Ext.get()和on()，当select的值发生改变时，就可以获得用户选择的主题名称，然后用Ext.util.CSS.swapStyleSheet改变主题样式。

(3) 给HTML添加一个空的CSS标签，以备后用。虽然这个标签现在是空的，但是在调用swap函数时它就会起作用了。

```
<link id="theme" rel="stylesheet" type="text/css" href="" />
```

示例在11.util/03-01.html中。

这里使用的还是HTML自带的下拉框select，相信很多朋友都想利用EXT自带的ComboBox来实现。调用swapStyleSheet()函数的部分不需要改变，只要创建一个用于选择主题的ComboBox即可，如下面的代码所示：

```
{
    id: 'themeSelect',
    hiddenName: 'comboId',
    xtype: 'combo',
    fieldLabel: '组合框',
    triggerAction: 'all',
```

```
        store: new Ext.data.SimpleStore({
            fields: ['value','text'],
            data: [
                ['default','默认风格'],
                ['gray','灰色空间']
            ]
        }),
        displayField:'text',
        valueField:'value',
        mode:'local',
        emptyText:'切换皮肤'
    }
```

这里使用的就是EXT的ComboBox来实现切换主题的功能，在第4章中已经讲解过它的使用方法。实现主题切换的代码如下所示：

```
//切换风格
Ext.getCmp('themeSelect').on('select', function(combo){
    if(combo.getValue()=='default'){
        Ext.util.CSS.swapStyleSheet("theme",
            "../../resources/ css/ext-all.css");
    }else{
        Ext.util.CSS.swapStyleSheet('theme',
            "../../resources/ css/xtheme-" + combo.getValue() + '.css');
    }
});
```

从上面的代码中可以看出，其实只有传入的参数发生了改变，传入的是ComboBox，而不是select下拉框。

示例在11.util/03-01.html中。

11.4 悬停提示

所谓悬停提示，就是当鼠标停在某个组件上方时会出现一个提示框，里面包含的内容是对这个组件的一些解释性描述。

EXT通过QuickTip实现了悬停提示功能，在表格的header部分、树形的node部分和表单的field部分都可以见到它。它不仅仅是弹出一个蓝色小框，我们还可以通过QuickTip为悬停提示定义其他的功能。

11

11.4.1 初始化

如果想使用提示功能，请务必先用Ext.QuickTips.init()函数进行初始化。只有在进行了初始化之后，我们才可以激活表格、树形、表单和自定义的tip，如下面的代码所示：

```
Ext.QuickTips.init();
```

在对EXT的提示体系进行初始化之后，就会立刻激活提示功能，必须在创建页面上其他组件之前对QuickTip进行初始化。Ext.QuickTips还包含一些其他操作，例如，用disable()函数禁用提示功能、用enable()函数启用提示功能、用register()函数为DOM元素注册提示功能，

以及用unregister()函数取消提示功能，所有这些操作都只有在初始化之后才能执行。当然，我们可能不需要执行这么多操作，现在只要先执行init()函数的操作即可。

11.4.2　注册提示

既然已经说到了register()注册函数，我们就来看看如何给一个DOM元素添加提示功能。

首先需要在页面中添加一个标签<input id="q1" type="button" value="提示信息"/>。现在所必需的只有它的id，其他的属性暂时与提示功能无关。

现在我们就为这个button注册提示信息，如下面的代码所示：

```
Ext.QuickTips.init();
Ext.QuickTips.register({
    target: 'q1',
    title: '第一型',
    text: '从外部注册到dom里的提示信息'
});
```

register()函数的最简形式只有3个参数：target指定给哪个DOM元素注册提示；title和text是显示在提示框中的信息，它们的区别是默认显示的位置和样式不同。默认情况下，title的值会被加粗并放到上面，text的值放在下面，我们还可以直接使用HTML来控制显示的格式。

设置完成后，只要把鼠标放到按钮上一会儿，就会出现这个提示了，效果如图11-13所示。



图11-13　无侵入提示

11.4.3　标签提示

上一节讨论了如何使用Ext.QuickTips.register()为DOM节点注册提示信息，现在来看一下如何直接在HTML中为元素设置提示信息。

请注意，即使将提示信息添加到HTML中，初始化过程也是必需的。现在我们就来看看如何在HTML中添加提示信息。

```
<input type="button" value="标签型提示信息"
    ext:qtitle="提示的标题"
    ext:qtip="<b><i>提示的内容</i></b><p>下一行内容。</p>" />
```

这些配置都以ext:作为前缀，qtitle代表提示标题，qtip代表提示内容，这些配置会在初始化时解析到EXT中，并在需要时显示。

11.4.4　全局配置

默认情况下，提示信息会在鼠标悬停0.5秒后显示，显示5秒后消失。有时需要修改它的默认配置，最简单的办法就是利用Ext.apply()函数重新设置QuickTips的默认属性，如下面的代码所示：

```
Ext.apply(Ext.QuickTips, {
    maxWidth: 200,
```

```
    minWidth: 100,
    showDelay: 50,
    dismissDelay: 1000,
    hideDelay: 500,
    trackMouse: false,
    animate: true
});
```

下面来看看这些设置的含义。

- maxWidth和minWidth分别用来设置提示框的最大宽度和最小宽度，这可能会导致提示内容自动换行。
- showDelay表示鼠标悬停多久后显示提示信息，默认是0.5秒，这里的时间以ms为单位。
- dismissDelay表示提示信息显示的时间，默认为5秒。如果希望提示框不消失，也可以将dismissDelay直接设置为0。
- hideDelay表示提示信息从开始消失到完全消失所需的时间，默认为0.2秒。
- trackMouse这个属性很有意思，如果把它设置为true，提示窗口弹出来以后还会跟随鼠标一起移动。
- autoHide的值一般都是true，它会根据dismissDelay和hideDelay的数值来实现逐渐消隐提示框的效果。这似乎也是一条让提示框永不消失的捷径，不过用过以后你就会知道，autoHide:false的结果是即使鼠标离开了DOM的范围，提示信息也不会消失。这个提示信息会一直挂在页面上，再也无法去掉了。
- animate表示是否使用动画效果。

11.4.5 个体配置

QuickTips中的这些参数既可以用于全局配置，也可以用于对某个组件进行单独指定，可以为某一个DOM的提示信息设置自己的配置。

单独配置有两种方式：使用register()和直接写到HTML里。在这两种方式中，配置名称稍有不同，但也有规律可循。

(1) 使用register()函数时，用到的参数和全局配置一样，如下面的代码所示：

```
Ext.QuickTips.register({
    target: 'q1',
    title: '第一种类型',
    text: '从外部注册到DOM中的提示信息',
    maxWidth: 200,
    minWidth: 100,
    showDelay: 50,
    dismissDelay: 1000,
    hideDelay: 500,
    trackMouse: false,
    autoHide: false,
    animate: true
});
```

这里的autoHide:false与之前讨论的有所不同，它单独设置autoHide:false，出现的提

示不会自动消失，但会给用户提供一个关闭按钮，全局设置中的参数不具备这项功能。

(2) 标签的写法不同，其实也可以写成一样的，或许Jack有其他的考虑吧。

```
<input type="button" value="自以为是标签型提示信息"
    ext:hide="user"
    ext:qclass=""
    ext:qwidth="300"
    ext:qtitle="标题"
    ext:qtip="<b><i>内容</i></b><p>直接写到标签里。</p>" />
```

这种标签式设置有着很大的局限，我们只能使用上述5个参数，其中`ext:hide="user"`的情况与单独配置中的`autoHide:false`的效果相同。其他参数都很容易理解，用于设置样式和大小。

示例在11.util/04-01-01.html中。

表11-2中列出了内置了提示功能的EXT组件，只需要在初始化QuickTips之后，再为这些组件设置对应的属性，就可以直接为它们设置对应的提示信息。

表11-2 内置了提示功能的组件和它们对应的配置

组件名称	配 置
Ext.tree.TreeNode	qtip
Ext.Button	tooltip

11.5 使用 Ext.state 保存状态

`Ext.state`提供了保存组件状态的功能，如果不了解它的内部实现，可能会遇到一些奇怪的问题。下面就来讨论一下如何使用`Ext.state`，以及如何避免使用过程中可能遇到的问题。

为了比较，我们先在页面中放一个被分成5个部分的Viewport布局和一个弹出窗口，实现过程如代码清单11-3所示。

代码清单11-3 设置Viewport布局和弹出窗口

```
Ext.onReady(function(){
    var viewport = new Ext.Viewport({
        layout:'border',
        items:[{
            region: 'north',
            height: 50,
            split: true
        },{
            region: 'south',
            height: 50,
            split: true
        },{
            region: 'west',
            width: 50,
            split: true
        },{
            region: 'east',
            width: 50,
```

```
                split: true
            },{
                region: 'center'
            }]
        });

        var win = new Ext.Window({
            title: 'title',
            width: 300,
            height: 200
        });
        win.show();

    });
```

在Viewport布局中，上下两部分的默认高度都是50（像素），左右两部分的默认宽度也是50（像素）。弹出窗口宽为300（像素），高为200（像素），默认情况下出现在浏览器中央，整体效果如图11-14所示。



图11-14 设置Viewport布局和弹出窗口

我们为Viewport布局的上、下、左、右4个部分都设置了split，这样就可以通过拖放来修改它们的宽度和高度了。弹出窗口既可以修改大小，又可以拖放，但是页面刷新后，窗口又会恢复原状。接下来，我们在页面中加入state，如下面的代码所示：

```
Ext.state.Manager.setProvider(new Ext.state.CookieProvider({}));
```

注意，这段代码应该添加到创建Viewport和Window之前。也就是说，对Ext.state.Manager的初始化功能必须放在所有代码的最前面，这和Ext.QuickTips的用法相似。

现在拖动窗口的位置，并调整页面布局，如图11-15所示。

然后刷新一下页面，结果布局和窗口的位置和大小都没有变化。是我们眼花了，还是刚才没有点中刷新按钮？把浏览器关掉，然后再打开这个页面看看。奇怪了，页面仍然没有任何变化。这是什么原因呢？有经验的朋友可能会认为是页面缓存的引起的，但是清除页面缓存后，页面仍然没有变化。其实这是Cookie引起的，如果不清空Cookie，初始化数据就不会起作用，页面布局就不会变。

Cookie里究竟藏着什么玄机？我们用FireFox来查看一下Cookie，如图11-16所示。



图11-15 调整Viewport布局并拖放窗口

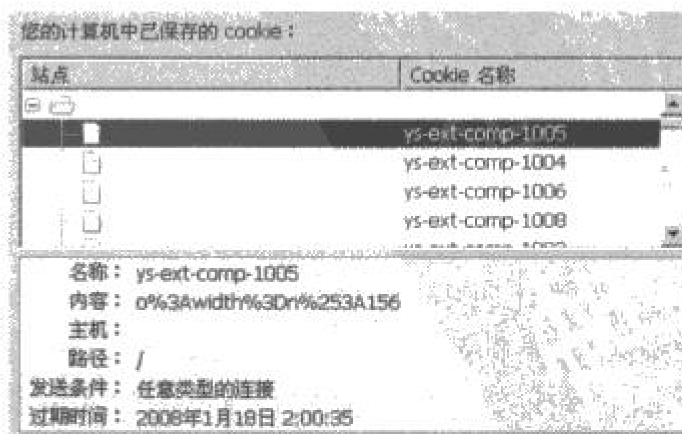


图11-16 Cookie中的秘密

这些以“ys”开头的文件就是EXT为我们保存的数据，先删除它们，然后再刷新一下，这次布局和窗口都回到原位了。

其实原理很简单，Ext.state是由Manager和Provider两部分组成的，Manager是一个单例，不需要每次都创建它，需要它时，直接在它里面设置Provider即可。这个Provider是用于保存和读取数据的功能组件，默认情况下EXT只为我们提供了一个Ext.state.CookieProvider，它会把组件的状态都保存到用户浏览器的Cookie里。默认情况下，这个Cookie会保存一周，Cookie的path和domain会被设置为当前网站的path和domain，因为我们刚才打开的网页是在本地的，所以这时设置的domain部分都为空值。

修改CookieProvider默认设置的方法，如下面的代码所示：

```
var cp = new Ext.state.CookieProvider({
    path: "/cgi-bin/",
    expires: new Date(new Date().getTime()+(1000*60*60*24*30)), //30天
    domain: "extjs.com"
});
Ext.state.Manager.setProvider(cp);
```

参数path和domain都是字符串，expires表示过期时间，是在当前时间上加上7天得到的。实际使用中，过期时间可以自行计算。

EXT在运行时，支持state的组件首先会判断Ext.state.Manager是否进行了初始化。如果有必要，就调用state中的数据对自己的属性进行修改。顺便提一句，所有Ext.Component都支持从state中获取状态。

我们已经看到了，在Cookie中保存的状态信息都是以“ys”开头的字符串，“ys”后面的部分是组件的id。而EXT组件的id很多都是自动生成的，既然是自动生成的，就可能会有重复，一旦出现重复，就会出错。可以想象一下，如果之前在测试页面中使用过CookieProvider，浏览器中就会保存一些组件的状态信息。如果之后又在其他页面里使用state，那么这个页面就会把上次保存的状态都取出来。如果正好遇到id相同的情况，就会出现问题。这个页面上创建的Window的id可能正好与上个页面中创建的ViewPort的id是相同的，这个页面上的Window就会从state中取出上次保存的宽度和高度，这样一来，显示结果就会错得很离谱。

为了避免这个问题，不应该让EXT自动为组件匹配state中的属性。我们推荐使用stateId参数，为这个参数设置一个不容易重复的名字，这样就不容易出现上述问题了。

```
{
    stateId: 'viewPortNorth',
    region: 'north',
    height: 50,
    split: true
}
```

上面的代码中，我们为组件配置了stateId:'viewPortNorth'，通过state保存到Cookie里的key就会变成ys-viewPortNorth，这样就不容易产生冲突了。

经过上面的讨论，我们了解了如何在实际工作中使用state保存组件的状态，也讨论了state可能造成的问题，以及避免这些问题的方法，大家可以在实际开发过程中酌情采用。

示例在11.util/05-01.html中。

11.6 使用 fx 实现的动画效果

如果打开EXT发布包中的sources目录，就会在core目录下看到一个名为fx.js的脚本文件。它是专门用来管理动画效果的脚本，不过EXT已经将这些动画效果合并到Element中了，我们可以直接在得到的el上调用fx.js中已经定义好的动画效果。

fx.js脚本中定义了多种动画效果，包括渐入渐出、突出显示、改变大小、闪烁等。每种动画效果都可以在对应函数的options参数中指定附加属性，其中duration表示持续时间，remove表示是否在消失后彻底删除DOM，callback代表在动画效果显示完毕后执行的回调函数，等等。

我们还可以通过anchor参数，指定在某个方向上实现动画效果，表11-3中列出了可选的参数值。

表11-3 anchor参数列表

数 值	说 明
tl	左上
t	上
tr	右上
l	左
r	右
bl	左下
b	下
br	右下

在指定了anchor参数之后，实现的动画效果会根据我们定义的方向展开，我们可以在这8个方向中任选其一。

fx.js中还包含了一些其他功能函数，比如pause()函数可以指定暂停几秒后执行下一个动画效果，syncFx()函数表示立即执行所有动画效果。

随书示例中演示了EXT所支持的全部动画效果，代码如下所示：

```
Ext.onReady(function(){
    var fx = Ext.get('fx01');
    var log = Ext.get('log');

    fx.pause(1);
    log.update('fadeOut()');
    fx.fadeOut({
        callback: function() {
            log.update('fadeIn()');
        }
    });

    fx.pause(1);
    fx.fadeIn({
        callback: function() {
            log.update('frame()');
        }
    });

    fx.pause(1);
    fx.frame('green', 1, {
        callback: function() {
            log.update('ghost()');
        }
    });
```

11

```
fx.pause(1);
fx.ghost('t', {
    callback: function() {
        log.update('highlight()');
        fx.show(true);
    }
});

fx.pause(1);
fx.highlight("0000ff", {
    callback: function() {
        log.update('puff()');
        fx.show(true);
    }
});

fx.pause(1);
fx.puff({
    remove: false,
    callback: function() {
        log.update('scale()');
        fx.show(true);
    }
});

fx.pause(1);
fx.scale(100, 100, {
    callback: function() {
        log.update('shift()');
    }
});

fx.pause(1);
fx.shift({
    callback: function() {
        log.update('slideOut()');
    }
});

fx.pause(1);
fx.slideOut('tl', {
    callback: function() {
        log.update('slideIn()');
    }
});

fx.pause(1);
fx.slideIn('tr', {
    callback: function() {
        log.update('switchOff()');
    }
});
```

```
        fx.pause(1);
        fx.switchOff({
            callback: function() {
                log.update('done...');
            }
        });


});
```

示例在11.util/06-01.html中。

11.7 局部更新网页内容

利用Ajax局部更新网页内容的知识在这里就不赘述了，我们来看一下EXT提供的方式。在EXT中，不但将这一连串的操作封装到了一起，而且还提供了许多高级功能。

先看一下页面上的效果，如图11-17所示。

按下第一个按钮，EXT就会通过Ajax读取一个静态页面的内容，把获得的页面内容显示到按钮下面的部分，如图11-18所示。

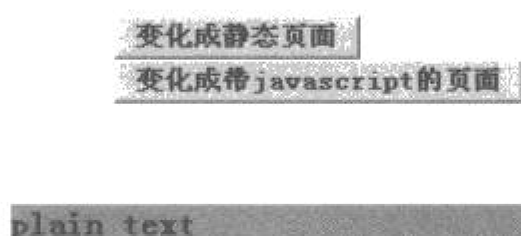


图11-17 更新成静态页面内容



图11-18 更新附带JavaScript的页面内容

按下第二个按钮，EXT就会去读取另一个静态页面的内容，把获得的页面内容显示在按钮下面的部分，同时还会执行页面里的JavaScript脚本，你将看到图11-18下面的长方形会显示动画效果。

这两种效果都是通过`Ext.Updater`实现的，EXT为我们提供的`Ext.Updater`不但可以更新页面内容，还可以设置是否执行页面中包含的JavaScript脚本。

现在我们来看一下代码，如果只想更新静态页面的内容，方法如下面的代码所示：

```
Ext.get('result').getUpdater().update({
    url: '07-02.html'
});
```

上面这段代码的执行过程如下所示。

(1) `Ext.get('result')`获得`id='result'`的`DIV`。

(2) `getUpdater()`获得`DIV`对应的`Ext.Updater`对象。

(3) 调用`update()`函数，取得`url:'07-02.html'`的内容，替换`DIV`的内容。

如果我们想执行后台HTML中的JavaScript代码，则要添加另一个参数，如下所示：

```
Ext.get('result').getUpdater().update({
```

```
        url: '07-03.html',
        scripts: true
    });
```

添加的一行参数是scripts:true，这样updater就会在获得页面数据之后自动检测数据中是否包含JavaScript脚本。如果页面中包含JavaScript，Ext.updater就会在读取页面内容后将页面中的JavaScript脚本提取出来并执行这些代码。

注意，如果获得的页面中夹杂了<script src="ext-all.js"></script>之类的标签片段，就会导致JavaScript脚本出错。因为updater会重新加载ext-all.js脚本，这会导致新加载的ext-all.js脚本与原来页面中的ext-all.js脚本发生冲突，导致整个页面停止响应。

示例在11.util/07-01.html中。

11.8 使用 Ext.util.Format 对数据进行格式化

顾名思义，Ext.util.Format的功能是将各种类型的数据转换为特定格式的字符串。在实际开发中，我们可以直接调用在它内部定义的常用工具函数来格式化数据。下面我们将对Ext.util.Format中的这些格式化函数进行分类介绍。

1. 操作字符串

- capitalize(String value)：将字符串开头的第一个字母变成大写，Ext.util.Format.capitalize('name');的结果就是Name。
- ellipsis(String value, Number length)：截取第一个参数value前面指定的多个字符，并在后面附加'...'，在文字排版时比较常用。
- htmlEncode(String value)：将文本编码，把其中的&、<、>转换成HTML中要求的编码形式"&"、"<"、">"。htmlDecode(String value)是htmlEncode()的逆操作。
- lowercase(String value)：将文本都转换成小写，对应的uppercase(String value)的功能是将文本都转换成大写。
- stripScripts(Mixed value)：删除文本中所有的<script>标签。
- stripTags(Mixed value)：删除文本中所有的标签。
- substr(String value, Number start, Number length)：截取子字符串。
- trim(String value)：去除文本两端的空白。

2. 操作日期

- date(Mixed value, [String format])：把日期变量转换成对应格式的字符串输出，Ext.util.Format.date(new Date(), 'Y-m-d')会得到类似于“2008-07-23”这样的字符串，它的结果是今天的日期。
- dateRenderer(String format)：这个函数可以看作专门为Ext.grid.GridPanel提供的工具函数，它会根据输入的日期格式返回一个renderer函数，之后Ext.grid.ColumnModel会利用这个renderer函数格式化表格中的日期。

3. 空值判断

- defaultValue(Mixed value, String defaultValue)：可以接收两个参数，如果第一个参数为空，就返回第二个参数的值，否则返回第一个参数的值。
- undef(Mixed value)：如果value为空，就返回空字符串，否则返回value的值。

4. 转换数字

- fileSize(Number/String size)：把数字和字符串转换成文件大小的形式，比如xxbytes、xxKB、xxMB。
- usMoney(Number/String value)：将数字转换为货币（美元）格式。

11.9 使用 Ext.util.CSS 管理 CSS 样式

Ext.util.CSS主要负责管理HTML页面中的CSS样式。如果我们希望将页面上的所有文字都设置成蓝色，可以使用如下代码：

```
Ext.util.CSS.createStyleSheet("x", "*{color:blue}");
```

这样会自动在页面的head标签中添加一个id="x"的标签，内容是*{color:blue}，页面中的所有元素会自动设置这段CSS，于是页面上所有的文字都变成了蓝色。

如果想知道当前已经设置了哪些CSS样式，我们可以使用getRule()和getRules()函数。getRules()函数会返回页面上已设置的所有CSS样式，而getRule()则会根据我们设置的CSS选择符返回对应的CSS样式，如下面的代码所示：

```
alert(Ext.util.CSS.getRules());
alert(Ext.util.CSS.getRule("*"));
```

在上例中，getRules()会返回所有的CSS样式，而getRule("*")返回的是CSS选择符*对应的样式。为了提高效率，第一次调用geRules()或getRule()函数时，EXT会自动将获得的结果缓存起来，以便下一次执行查找操作。但如果动态添加了CSS样式，EXT并不会自动刷新查找结果，这时就需要我们调用refreshCache()函数告知EXT刷新缓存，也可以直接在调用getRule()或getRules()函数时使用refreshCache参数来达到刷新缓存的目的，如下面的代码所示：

```
Ext.util.CSS.refreshCache();
Ext.util.CSS.getRule("*", true);
Ext.util.CSS.getRule(true);
```

对于已存在的CSS样式，可以使用updateStyleSheet()函数对指定的样式进行更新。比如我们希望将上面例子中设置的蓝色字体修改为红色字体，可以使用如下代码：

```
Ext.util.CSS.updateRule("*", "color", "red");
```

也可以删除指定id的style标签，页面上对应元素的样式会恢复到设置标签之前的状态，如下面的代码所示：

```
Ext.util.CSS.removeStyleSheet("x");
```

经过这几步操作之后，我们删除了之前设置创建的style标签，页面上的字体也就恢复到添加标签之前的颜色了。

11.10　使用 **Ext.util.ClickRepeater** 处理点击事件

Ext.util.ClickRepeater监听页面上任意一个元素的点击事件，在用户一直按住鼠标时，ClickRepeater会每隔一段时间重复触发一次click事件，以此来模拟用户连续点击某个元素的场景。

下面我们来演示一个示例，只要点击一次按钮，并持续按住鼠标，就会不断触发click事件，在页面上显示越来越多的字符，如图11-19所示。

```
标题
内容
click button
click click click click click click click
click click click click click click click
click click click click click click click
click click click click click click click
click click click click click click click
click click click click click click click
click click click click click click click
click click click click click click click
click click click click click click click
click click click click click click click
click click click click click click click
click click click click click click click
```

图11-19　使用ClickRepeater实现重复点击的功能

上例使用ClickRepeater监听click button的点击事件，并在每次重复触发click事件时向HTML中写入click内容，代码如下所示：

```
var clickRepeater = new Ext.util.ClickRepeater(Ext.get('button'));
clickRepeater.on('click', function() {
    Ext.getDom('clickResult').innerHTML += ' click'
});
```

事实上，我们只需要先根据id获得一个Element实例，然后用这个Element实例创建clickRepeater对象，之后只需要为这个clickRepeater对象设置click事件的监听函数就可以了。在上例的监听函数中，只是不断地向clickResult这个DOM内添加'click'字符串。最终页面上显示的效果就像图11-19一样，当我们单击鼠标时，只要不放开鼠标键就会连续触发click事件，页面上的“click”会不断变多。

ClickRepeater除了可以实现连续点击事件以外，还提供了一些附加功能，诸如元素被点击时显示的样式，发生连续点击时每次点击的时间间隔都可以自行设置，如下面的代码所示：

```
var clickRepeater = new Ext.util.ClickRepeater(Ext.get('button'), {
    delay: 1000,
    interval: 400,
    pressClass: 'pressButton'
});
clickRepeater.on('click', function() {
```

```
    Ext.getDom('clickResult').innerHTML += ' click'
});
```

我们使用ClickRepeater构造函数的第二个参数设置附加功能，上例中设置了delay、interval和pressClass这3个参数。

- delay：1000表示用户按住鼠标后，1000 ms之后开始执行连续点击。
- interval：400表示开始执行连续点击后，每次点击之间的间隔是400 ms。
- pressClass：'pressClass'表示用户按下鼠标后使用的CSS样式，该样式会在用户松开鼠标后自动从当前元素上移除。

上面代码的显示效果如图11-20所示。



图11-20 为ClickRepeater设置附加功能参数

示例放在11.util/10-01.html中。

11.11 使用 Ext.util.DelayedTask 延时执行函数

Ext.util.DelayedTask可以延迟执行某一段功能函数，它的作用与JavaScript中提供的setTimeout()函数有些相似。下面我们来讨论一下如何在实际中使用Ext.util.DelayedTask。

```
Ext.get('button').on('click', function() {
    var delay = new Ext.util.DelayedTask(function() {
        var text = new Date().toLocaleString() + "<br />";
        Ext.get('result').update(text);
    });
    delay.delay(500);
});
```

上例中，每次单击button都会创建一个Ext.util.DelayedTask实例，它会将当前时间显示到id="result"的DOM内部。在创建delay实例之后就调用它的delay(500)函数，这会使定义在DelayedTask中的回调函数延迟500 ms后再执行。因为在单击button之后，再过500 ms之后才能在页面上看到显示当前时间。

如果我们希望在回调函数执行前取消操作，可以调用cancel()函数。

如果延迟执行的回调函数需要设置范围或参数，我们也可以在创建DelayedTask时进行设置，如下面的代码所示：

```
App = {
    msg: ' Hello: ',
    recall: function(name, title) {
        var text = new Date().toLocaleString()+ this.msg + title + ' ' + name + "<br />";
        Ext.get('result').update(text);
    }
};
Ext.get('button').on('click', function() {
    var delay = new Ext.util.DelayedTask(App.recall, App, ['kayzhan', 'Ms.']);
    delay.delay(500);
});
```

DelayedTask构造函数的第二个参数为延迟执行的回调函数指定scope，第三个参数为延迟

11

执行的回调函数指定执行时传入的参数列表。请注意，这里传递的参数列表是一个数组。

DelayedTask只适用于对某一功能函数延迟执行一次的情况，如果希望每隔一段时间就执行一次功能函数，就会发现它所控制的延迟时间是非常不精确的，如下面的代码所示：

```
var time_display = "";
function recall() {
    var t = new Date();
    time_display += t.toLocaleString() + "<br>";
    viewport.items.itemAt(0).body.update(time_display);
    delay.delay(3000);
}
var delay = new Ext.util.DelayedTask(function(){recall();});
delay.delay(3000);
```

上述代码中，我们在回调函数recall()执行完毕后再次执行了delay(3000)，希望3秒后能够再次执行回调函数。但是，多次调用之后就会出现延时错误的问题，如图11-21所示。

```
2008年12月8日 11:15:15
2008年12月8日 11:15:18
2008年12月8日 11:15:21
2008年12月8日 11:15:27
2008年12月8日 11:15:30
2008年12月8日 11:15:33
```

图11-21　多次执行DelayedTask后出现延时错误

可以看到11:15:21到11:15:27之间相差了6秒，而不是我们期望的3秒，这是由于DelayedTask内部对数值计算不精确造成的。对于循环执行的任务，我们应该使用下面介绍的TaskRunner，而不是直接使用DelayedTask。

示例在11.util/11-01.html中。

11.12　使用 **Ext.util.TaskRunner** 执行循环任务

Ext.util.TaskRunner可以管理一系列回调函数，让它们以并行的方式循环执行。它提供了start()、stop()和StopAll()等方法，用来控制功能函数的启动和停止，或一次性停止所有已经执行的功能函数。不过在使用Ext.util.Runner之前，首先需要按照要求创建一个JSON对象，这个JSON对象中需要包含回调函数run()和循环间隔interval，之后再调用TaskRunner的start()函数，就可以启动整个任务了，如下面的代码所示：

```
var task = {
    run: function() {
        Ext.get('result').update(new Date().toLocaleString());
    },
    interval: 1000
};
var taskRunner = new Ext.util.TaskRunner();
taskRunner.start(task);
```

上例中，我们每隔1秒就更新一次id="result"内部的时间信息，就像实现了一个时钟一样。如果希望停止执行任务，可以调用stop()函数停止这个任务，或使用stopAll()函数停止当前所有正在运行的任务，如下面的代码所示：

```
Ext.get('stop').on('click', function() {
    taskRunner.stop(task);
});
Ext.get('stopAll').on('click', function() {
    taskRunner.stopAll();
});
```

我们将DelayedTask中出现错误的例子放到TaskRunner中执行，如下面的代码所示：

```
var text = '';
var task = {
    run: function() {
        text += new Date().toLocaleString() + "<br />";
        Ext.get('result').update(text);
    },
    interval: 3000
};
var taskRunner = new Ext.util.TaskRunner();
taskRunner.start(task);
```

我们还是每隔3秒向页面输出一次当前时间，从页面上的显示结果来看，没有发生延时错误的情况，说明使用TaskRunner执行定时任务是值得信任的，如图11-22所示。

button
2008年12月8日 11:38:06
2008年12月8日 11:38:09
2008年12月8日 11:38:12
2008年12月8日 11:38:15
2008年12月8日 11:38:18
2008年12月8日 11:38:21
2008年12月8日 11:38:24
2008年12月8日 11:38:27
2008年12月8日 11:38:30
2008年12月8日 11:38:33
2008年12月8日 11:38:36
2008年12月8日 11:38:39
2008年12月8日 11:38:42
2008年12月8日 11:38:45
2008年12月8日 11:38:48
2008年12月8日 11:38:51
2008年12月8日 11:38:54
2008年12月8日 11:38:57
2008年12月8日 11:39:00

图11-22 使用TaskRunner执行定时任务

示例在11.util/12-01.html中。

11.13 混合型集合 Ext.util.MixedCollection

Ext.util.MixedCollection是一个混合型集合类，它既可以支持用数字型索引获取和保存数据，也可以支持key:value对的形式获取和保存数据。MixedCollection不但提供了与集合相关的功能函数，还提供了对事件的支持。我们可以监听这些事件以了解外部对MixedCollection执行的操作。

1. MixedCollection的基本操作

我们先使用MixedCollection完成集合中最基本的操作（添加、删除、查找和修改），如下面的代码所示：

```
var collection = new Ext.util.MixedCollection();
//
collection.add(1);
collection.add(2);
collection.add(3);
//
var result = [];
for (var i = 0; i < collection.getCount(); i++) {
    result.push(collection.get(i));
}
alert(result);
//
```

```
collection.removeAt(0);
//
collection.replace(0, 200);
//
result = [];
for (var i = 0; i < collection.getCount(); i++) {
    result.push(collection.get(i));
}

alert(result);
```

上例创建了一个Ext.util.MixedCollection实例，调用它的add()函数向实例中添加了几个数值，然后使用for循环将collection中的数值读取出来放入一个数组中，再输出这个数组中包含的值，得到的结果是[1,2,3]。

接下来，我们调用removeAt()函数，将collection中的第一个数值删除，然后调用replace(0,200)将collection中的第一个数值替换为200，再次使用for循环读取collection中的数值并放入一个数组中，这次输出的结果是[200,3]。

2. 向MixedCollection中添加数据

MixedCollection提供了多种向集合中添加数据的方式，我们可以使用add()函数一次只向集合中添加一条数据，也可以使用addAll()函数将一个数组或一个JSON对象中的数据依次添加到集合中。MixedCollection还提供了insert()函数，允许用户指定添加数据的位置。这几种添加数据的方法如下所示：

```
var collection = new Ext.util.MixedCollection();
//
collection.add(1);
collection.addAll([2, 3]);
collection.insert(1, 100);
```

上述代码先向collection中添加了一条数据1，然后再将数组中的两条数据添加到collection中，最后将100插入到collection的索引为2的位置，这样得到的结果就是[1,100, 2,3]。

3. 删除MixedCollection中的数据

如果需要删除MixedCollection中的数据，可以选择使用remove()或removeAt()函数从集合中删除一条数据，或使用clear()函数删除集合中的所有数据，如下面的代码所示：

```
collection.removeAt(0);
collection.remove("3");
//
collection.clear();
```

removeAt()与remove()之间的区别是，removeAt()的参数是索引值，它会删除集合中指定索引值的数据，remove()的参数是我们希望删除的数据，它会将集合中与参数值相同的数据删除。Clear()函数的作用是清空整个集合，它会删除集合中所有的数据。

4. 修改MixedCollection中的数据

MixedCollection只提供了一个修改数据的函数replace()，我们需要为replace()指定

需要修改的数据在集合中所处的位置和修改之后的数据，如下面的代码所示：

```
collection.replace(2, "xxx");
```

上述代码将会把集合内的第三个数据修改为"xxx"。

5. 读取MixedCollection中的数据

在了解了如何添加、删除和修改MixedCollection内的数据之后，我们还需要执行读取数据的操作。MixedCollection提供了一系列读取内部数据的函数，如下面的代码所示：

```
var collection = new Ext.util.MixedCollection();
collection.addAll(["11", "22", "33", "44", "55"]);
var result = Ext.getDom('result');
result.innerHTML += collection.first() + "<br />";
result.innerHTML += collection.last() + "<br />";
result.innerHTML += collection.getCount() + "<br />";
result.innerHTML += collection.get(1) + "<br />";
result.innerHTML += collection.indexOf("33") + "<br />";
```

上述代码列出了从MixedCollection中获取数据的最基本的几种方法。first()和last()函数分别获得集合中的第一条和最后一条数据；getCount()函数得到的是集合中的数据的数量；get()函数根据索引值返回集合中对应的数据；indexOf()则是返回集合中数据对应的的索引值。通过上面这些功能函数，我们已经能够对集合中的数据进行常规的读取操作了。

6. 对MixedCollection中的数据执行复杂的查询操作

如果我们需要执行复杂的条件查询，可以使用MixedCollection提供的find()系列函数，如下所示：

```
var collection = new Ext.util.MixedCollection();
collection.addAll([{name:"11"}, {name:"22"}, {name:"33"}, {name:"44"}, {name:"55"}]);
var result = Ext.getDom('result');
result.innerHTML += collection.find(function(o) {
    return o.name == '11';
}) + "<br />";
result.innerHTML += collection.findIndex("name", "1") + "<br />";
result.innerHTML += collection.findIndexBy(function(o) {
    return o.name == '11';
}) + "<br />";
```

find()函数支持使用回调函数判断集合中的对象是否满足查询要求，如果存在满足查询条件的对象，它会返回一个满足条件的对象。FindIndex()函数会对集合中的对象的某个属性进行匹配，并返回第一个满足条件的对象的索引值，findIndexBy()函数使用回调函数查询集合中的对象，并返回第一个满足条件的对象的索引值。通过find系列的这3个函数，我们可以实现对集合内的数据执行复杂的查询操作。

7. 复制MixedCollection中的数据

可以使用clone()函数复制一个与原有实例内容完全相同的MixedCollection对象，也可以使用filter()或filterBy()函数将集合中符合要求的部分复制到一个新的MixedCollection对象中，如下面的代码所示：

```
var collection = new Ext.util.MixedCollection();
collection.addAll([{name:"11"}, {name:"22"}, {name:"33"}, {name:"aa"}, {name:"bb"}]);
var collection1 = collection.clone();
var collection2 = collection.filter("name", /^\d+$/);
var collection3 = collection.filterBy(function(o) {
    return /^\D+$/.test(o.name);
});
Ext.getDom('result').innerHTML += collection1.getCount() + "<br />";
Ext.getDom('result').innerHTML += collection2.getCount() + "<br />";
Ext.getDom('result').innerHTML += collection3.getCount() + "<br />";
```

使用clone()函数生成的MixedCollection实例中有5条数据，因为它将原MixedCollection中的数据都复制到了新实例中。而filter()和filterBy()返回的实例中分别只有3条和2条记录，因为filter('name',/^\d+$/)使用正则表达式只保留集合中name属性为数字的数据，而filterBy()函数只保留集合中name属性不为数字的数据，这样我们就可以实现根据条件复制MixedCollection中的数据的功能了。

8. 使用key:value的方式操作MixedCollection中的数据

MixedCollection还支持key:value的操作方式，可以为每条记录设置对象的key值，之后就可以使用key值操作集合内的数据。使用key值操作的方法与使用索引操作的方法类似，如下面的代码所示：

```
var collection = new Ext.util.MixedCollection();
collection.add("key1", 1);
collection.add("key2", 2);
collection.add("key3", 3);
collection.insert(1, "key10", 100);
var result = [];
for (var i = 0; i < collection.getCount(); i++) {
    result.push(collection.get(i));
}
alert(result);
collection.remove("key3");
collection.replace("key2", 200);
result = [];
for (var i = 0; i < collection.getCount(); i++) {
    result.push(collection.get(i));
}
alert(result);
```

上述代码示例中，我们在向集合中添加数据时全部使用了key:value的形式，为每个添加的数据都指定了对应的key值，这样我们就可以在删除和修改这些数据时直接使用key值了。

9. MixedCollection中的事件

MixedCollection继承了Observable，因此可以为它设置监听函数，处理其内部触发的各种事件。MixedCollection内部定义了add、clear、remove、replace等4个事件，分别对应集合中的添加数据、清空数据、删除数据和更新数据等操作。可以为这4个事件设置监听函数，从而获知集合内部数据的变化，如下面的代码所示：

```
var collection = new Ext.util.MixedCollection();
```

```
collection.on('add', function(index, o, key) {
    alert("在" + index + "添加了数据" + o + ", key为" + key);
});
collection.on('clear', function() {
    alert("集合数据被清空");
});
collection.on('remove', function(o, key) {
    alert("删除了数据" + o + ", key为" + key);
});
collection.on('replace', function(key, oldObject, newObject) {
    alert("修改了key为" + key + "的数据" + newObject + ", 修改前的值为" + oldObject);
});

Ext.get('add').on('click', function() {
    collection.add(new Date().getTime());
});
Ext.get('clear').on('click', function() {
    collection.clear();
});
Ext.get('remove').on('click', function() {
    collection.removeAt(0);
});
Ext.get('replace').on('click', function() {
    collection.replace(0, new Date().toLocaleString());
});
```

当我们对集合内的数据执行操作时，就会自动触发事件，执行监听函数内的操作。

示例在11.util/13-01.html中。

11.14 使用 Ext.util.TextMetrices 获得文本所占的高度和宽度

在我们希望获得页面上文本所占的高度和宽度时，就需要使用Ext.util.TextMetrices，如下面的代码所示：

```
var metrics = Ext.util.TextMetrics.createInstance("text");
var size = metrics.getSize("www.family168.com");
Ext.getDom('result').innerHTML += size.width + "," + size.height;
```

首先，我们通过Ext.util.TextMetrics类的createInstance()函数创建一个实例。然后，调用实例的getSize()函数，EXT会自动获得id="text"这个标签上定义的CSS样式，并以此作为计算文本大小的依据，我们得到的size中就包含了文本的实际宽度和高度。最后显示出的结果是136 px和18 px。这说明，如果把www.family168.com这段字符串显示在id="text"标签中，它的宽度是136 px，高度是18 px。

对于已创建完成的metrics实例，如果我们希望改变与文本显示相关的CSS样式，可以使用bind()函数，它的参数是HTML上的一个标签，它会将这个标签上的CSS样式复制过来，用作下一次计算文本宽度和高度的依据，如下面的代码所示：

```
metrics.bind("result");
var size = metrics.getSize("www.family168.com");
Ext.getDom('result').innerHTML += size.width + "," + size.height + "<br />";
```

11

在HTML中，id="result"的标签上设置了font-weight:bold的CSS样式，metrics实例将这段CSS样式复制之后，后面的文字都会以粗体形式进行计算，所以最后得到的宽度为153 px，高度为18 px。

对于div标签这种可能出现自动换行的情况，metrics提供了setFixedWidth()函数。事先使用setFixedWidth()函数为文本设置一个固定的宽度，然后调用getSize()函数就可以获得自动换行后的实际高度和宽度，如下面的代码所示：

```
var metrics = Ext.util.TextMetrics.createInstance("result");
metrics.setFixedWidth(100);
size = metrics.getSize("《深入浅出Ext JS》family168出品");
Ext.getDom('result').innerHTML += size.width + "," + size.height;
var size = metrics.getSize("www.family168.com");
Ext.getDom('result').innerHTML += size.width + "," + size.height;
```

上述代码中，使用setFixedWidth(100)将最大宽度设置为100px。计算"《深入浅出Ext JS》family168出品"这段文本将会获得100 px和72 px的结果，这说明文字如预期的一样换行显示了。但是，对于"www.family168.com"这段文字，获得的结果却是100 px和18 px。因为HTML中无法自动将一长串英文单词截断，所以，虽然文字已经超出了设置的最大宽度，但也无法产生自动换行的效果。这种情况下，Ext.util.TextMetrices也就无法自动计算出文字实际占用的宽度和高度了。实际情况中，这段文字有可能与其他部分的文字重复。

示例在11.util/14-01.html中。

11.15　使用 Ext.KeyNav 处理导航按键

Ext.KeyNav可以为某一对象绑定导航按键，导航按键包括enter、left、right、up、down、tab、esc、pageUp、pageDown、del、home和end等12个按键。

Ext.KeyNav可以绑定在任意一个对象上，如下面的代码所示：

```
var el = Ext.get('textarea');
var keyNav = new Ext.KeyNav(el, {
    left: function(e) {
        el.setWidth(el.getWidth() - 10);
    },
    right: function(e) {
        el.setWidth(el.getWidth() + 10);
    },
    up: function(e) {
        el.setHeight(el.getHeight() - 10);
    },
    down: function(e) {
        el.setHeight(el.getHeight() + 10);
    }
});
```

我们将一个KeyNav绑定在文本域（TextArea）上，当用户在文本域上按下left键和right键时，会改变文本域的宽度；当用户在文本域上按下up键和down键时，会改变文本域的高度。每次按键都会在对应的宽度或高度上增减10个像素。

在为一个对象绑定了KeyNav之后，如果希望取消响应按键功能，可以调用KeyNav提供的disable()函数，这时使用KeyNav在该对象上定义的所有按键功能都将被取消。在需要启用KeyNav的功能时，随时都可以调用enable()函数激活KeyNav，这时该对象上由KeyNav定义的所有按键功能都将被启用，对KeyNav禁用和启用的功能不会影响对原来对象的正常操作。禁用和启用功能如下面的代码所示：

```
Ext.get("dis").on('click', function() {
    keyNav.disable();
});
Ext.get('en').on('click', function() {
    keyNav.enable();
});
```

这样做，点击id="dis"按钮会将keyNav禁用，我们再也无法使用按键控制textarea改变大小了。点击id="en"按钮会将keyNav启用，之后又可以在textarea上使用定义好的功能函数了。

示例在11.util/15.html中。

11.16 使用 Ext.KeyMap 为对象绑定按键功能

在Ext.KeyNav中我们只能处理12个按键，在希望处理更多按键时，KeyNav就显得力不从心了，这时我们需要的是Ext.KeyMap，它对键盘上的每个按键都做了映射，可以使用它为任意一个按键设置处理函数，代码如下所示：

```
var keyMap = new Ext.KeyMap('textarea', {
    key: Ext.EventObject.LEFT,
    fn: function(e) {
        keyMap.el.setWidth(keyMap.el.getWidth() - 10);
    }
});
```

Ext.KeyMap和Ext.KeyNav的使用方法有些相似，不同的是，在Ext.KeyMap中可以直接使用key指定需要监听的按键。这里key对应的可以是任意一个按键，在绑定的对象上发生对应按键按下的事件时，就会触发fn对应的回调函数，这里使用的回调函数与Ext.KeyNav有些不同。在Ext.KeyNav中，如果不指定scope，那么回调函数中的this引用默认会指向我们所创建的Ext.KeyNav实例。但是，在Ext.KeyMap中，如果不指定scope，那么回调函数中的this引用默认会指向当前页面的window对象。

Ext.KeyMap不仅拥有与Ext.KeyNav一样的disable()和enable()函数，还可以对按键处理函数执行禁用和启用操作。它还提供了一个isEnabled()函数，通过这个函数我们可以得知keyMap实例的状态。在keyMap的状态为启用时，isEnabled()函数返回true；在keyMap的状态为禁用时，isEnabled()函数返回false，如下面的代码所示：

```
Ext.get("dis").on('click', function() {
    keyMap.disable();
    Ext.get('result').update(keyMap.isEnabled());
});
Ext.get('en').on('click', function() {
```

```
    keyMap.enable();
    Ext.get('result').update(keyMap.isEnabled());
});
```

Ext.KeyMap也可以一次绑定多个按键事件，如下面的代码所示：

```
var keyMap = new Ext.KeyMap('textarea', [{
    key: Ext.EventObject.LEFT,
    fn: function(e) {
        keyMap.el.setWidth(keyMap.el.getWidth() - 10);
    }
},{
    key: Ext.EventObject.RIGHT,
    fn: function(e) {
        keyMap.el.setWidth(keyMap.el.getWidth() + 10);
    }
},{
    key: Ext.EventObject.UP,
    fn: function(e) {
        keyMap.el.setHeight(keyMap.el.getHeight() - 10);
    }
},{
    key: Ext.EventObject.DOWN,
    fn: function(e) {
        keyMap.el.setHeight(keyMap.el.getHeight() + 10);
    }
}]);
```

上例中我们使用数组作为参数，将left、right、up、down这4个按键事件都绑定在textarea这个对象上，每个按键各自定义了对应的处理函数，这样我们就可以使用按键改变textarea的大小了。

Ext.KeyMap支持一次为多个按键事件设置同一个监听器。下例中，在用户按下a、b、c、d这4个键中的任意一个键时，都会将textarea的背景变为红色，并在1秒后恢复为白色。在设置key值时直接使用'abcd'的形式进行赋值，如下面的代码所示：

```
var keyMap = new Ext.KeyMap('textarea', {
    key: 'abcd',
    fn: function(e) {
        keyMap.el.setStyle("backgroundColor", "red");
        var fn = function(){
            keyMap.el.setStyle("backgroundColor", "white");
        };
        fn.defer(1000);
    }
});
```

对于不能使用字符表示的按键事件，也可以使用数组的方式定义key的值，如下面的代码所示：

```
var keyMap = new Ext.KeyMap('textarea', {
    key:[Ext.EventObject.ENTER,Ext.EventObject.BACKSPACE,Ext.EventObject.SPACE],
    fn: function(e) {
        keyMap.el.setStyle("backgroundColor", "red");
```

```
            var fn = function(){
                keyMap.el.setStyle("backgroundColor", "white");
            };
            fn.defer(1000);
        }
    });
```

上例使用数组的形式为key设置了3个元素，textarea会在用户按下回车、退格或空格键时执行事件处理函数。

Ext.KeyMap还支持对组合按键的处理，比如可以要求：只有在按下回车键后，再按下向左、向右、向上和向下键才能修改textarea的大小，如下面的代码所示。

```
var keyMap = new Ext.KeyMap('textarea', [{
    key: Ext.EventObject.LEFT,
    ctrl: true,
    fn: function(e) {
        keyMap.el.setWidth(keyMap.el.getWidth() - 10);
    }
},{
    key: Ext.EventObject.RIGHT,
    ctrl: true,
    fn: function(e) {
        keyMap.el.setWidth(keyMap.el.getWidth() + 10);
    }
},{
    key: Ext.EventObject.UP,
    ctrl: true,
    fn: function(e) {
        keyMap.el.setHeight(keyMap.el.getHeight() - 10);
    }
},{
    key: Ext.EventObject.DOWN,
    ctrl: true,
    fn: function(e) {
        keyMap.el.setHeight(keyMap.el.getHeight() + 10);
    }
}]);
```

示例在11.util/16-01.html中。

11.17 扩展

EXT还对JavaScript中的原生类型进行了扩展，为这些原生类型添加了附加功能，但并没有修改类型中的原有功能，所以不会影响我们编写的JavaScript函数。

11.17.1 扩展 Date

Date是EXT中扩展功能最丰富的类型，EXT为Date添加了许多工具函数。例如，getFirstDateInMonth()函数可以得到当前月份第一天的日期，getLastDateOfMonth()函数可以得到当前月份最后一天的日期。

EXT还为Date类型提供了与闰年相关的函数：isLeapYear()可以判断当前年份是否为闰年；getDayOfYear()可以获得当前日期是这一年的第几天；getDaysInMonth()可以获得当前日期是这一个月的第几天；getWeekOfYear()可以获得当前日期是这一年的第几周。

add(String interval, Number value)函数用于计算日期，这个函数可以在原有日期的基础上增加或减少某一项的数值，如下所示：

```
var dt = new Date('10/29/2006').add(Date.DAY, 5);
```

这是在原有日期的基础上增加5天，dt是我们获得的新的日期变量，它的值应该是'11/3/2006'。除了Date.DAY之外，我们可以使用Date.HOUR、Date.MILLI、Date.MINUTE、Date.MONTH、Date.SECOND和Date.YEAR来指定需要增加和减少的项目。

我们可以使用函数between()和getElapsed()来判断日期范围。between(Date start, Date end)可以判断日期变量是否在给定的范围之内，getElapsed([Date date])则获得日期变量与指定日期之间相差的毫秒数，它可以用于计算两个时间点之间的间隔。

在EXT中，日期类型的格式化功能也有大幅增强，可以直接使用format('Y-m-d');获得特定格式的日期，也可以通过parse('2008-07-24', 'Y-m-d');将字符串解析为对应的日期变量。

表11-4列出了可以选择的日期格式符号。

表11-4　日期格式符号

格　式	说　明	示　例
d	日期，两位数字，不足时补零	01 ~ 31
D	星期的简写	Mon ~ Sun
j	日期，不会补零	1 ~ 31
l	星期的全称	Sunday ~ Saturday
N	使用数字表示星期(ISO-8601)	1 (Monday) ~ 7 (Sunday)
S	日期后缀，与j配合使用	St、nd、rd或th
w	使用数字表示星期	0 (Sunday) ~ 6 (Saturday)
z	一年中的第几天（从0开始）	0 ~ 364 (闰年365)
W	一年中的第几个星期（ISO-8601从周一开始）	
F	月份的全称	January ~ December
m	月份，两位数字，不足时补零	01 ~ 12
M	月份的简写	Jan ~ Dec
n	月份，不会补零	1 ~ 12
t	当前月份一共有多少天	28 ~ 31
L	是否为闰年	闰年为1，否则为0
o	年份（ISO-8601）与Y相同，但是如果ISO星期（W）属于去年或明年，就使用它来表示今年	1998，2004
Y	年份，4位数字	1999，2003
y	年份，2位数字	99，03
a	上午，下午（小写）	am，pm
A	上午，下午（大写）	AM，PM

（续）

格 式	说 明	示 例
g	小时（12时制），不会补零	1 ~ 12
G	小时（24时制），不会补零	0 ~ 23
h	小时（12时制），不足时补零	01 ~ 12
H	小时（24时制），不足时补零	00 ~ 23
i	分钟，不足时补零	00 ~ 59
s	秒钟，不足时补零	00 ~ 59
u	毫秒，不足时补零	001 ~ 999
O	用小时和分钟表示与GMT的差异	+1030
P	用小时和分钟表示与GMT的差异，带冒号	-08:00
T	当前系统设定的时区	EST、MDT、PDT……
Z	用秒数表示的时区偏移量（西方为负数，东方为正数）	-43200 ~ 50400
c	ISO-8601的日期格式	2007-04-17T15:19:21+08:00，2007-04-17T15:19:21Z
U	与Unix Epoch（January 1 1970 00:00:00 GMT）相差的秒数	1193432466，-2138434463

11.17.2 扩展 String

EXT为字符串类型提供了几个工具函数，如下所示。

- escape(String string)：它会对字符串中的“'”和“\”进行转义处理。
- format(String string, String value1, String value2)：它为我们提供了一个简单的自定义模板，第一个参数传递一个包含替换标志的字符串，后面的参数会根据替换标志放到对应的位置上，如下所示。

```
var cls = 'my-class', text = 'Some text';
var s = String.format('<div class="{0}">{1}</div>', cls, text);
```

  得到的s值将是'<div class="my-class">Some text</div>'。
- leftPad(String string, Number size, [String char])：它保证string的长度不能小于size的值。如果不够，就在左侧使用第三个参数char指定的字符补齐，如下面的代码所示。

```
var s = String.leftPad('123', 5, '0');
```

  上面代码中得到的s值将是'00123'。
- toggle(String value, String other)：如果当前字符串与第一个参数相同，就返回第二个参数，否则返回第一个参数，如下面的代码所示。

```
sort = sort.toggle('ASC', 'DESC');
sort = (sort == 'ASC' ? 'DESC' : 'ASC');
```

  以上两条语句的效果是相同的，当sort与'ASC'相同时就返回'DESC'，否则返回'ASC'。
  trim()清空字符串两侧空白，如下面的代码所示。

```
var s = '  foo bar  ';
```

```
alert('-' + s + '-');          //获得 "-  foo bar  -"
alert('-' + s.trim() + '-');   //获得 "-foo bar-"
```

11.17.3 扩展 Function

Function是JavaScript中的一个默认类型，所有的函数都是它的实例。EXT扩展了Function类型，为JavaScript中的所有函数增加了扩展功能。

createCallback()函数会创建当前函数的回调函数，如下面的代码所示。

```
var sayHi = function(name){
    alert('Hi, ' + name);
}

new Ext.Button({
    text: 'Say Hi',
    renderTo: Ext.getBody(),
    handler: sayHi.createCallback('Fred')
});
```

单击示例中的按钮，就会弹出"Hi, Fred"的提示框，createCallback()的作用是为原有的参数设置默认参数。像上例中那样，在使用createCallback()时就已经将'Fred'设置给对应的回调函数，单击按钮后就会将参数传递给sayHi()。

createDelegate([Object obj], [Array args], [Boolean/Number appendArgs])函数会创建当前函数的代理函数，如下面的代码所示。

```
var sayHi = function(name){
    alert('Hi, ' + name + '. You clicked the "' + this.text + '" button.');
}

var btn = new Ext.Button({
    text: 'Say Hi',
    renderTo: Ext.getBody()
});

btn.on('click', sayHi.createDelegate(btn, ['Fred']));
```

createDelegate()用来为当前函数设置代理，主要功能是改写函数中的this引用。示例中createDelegate()的第一个参数是btn，sayHi()函数中的this.text会引用btn.text，获得它的数值'Say Hi'。如果将btn改为其他对象，sayHi()函数中的this会自动引用到指定对象的text属性。

createDelegate()还可以使用数组的形式为被代理函数设置默认传递的参数。这种用法与上面的createCallback()有些类似，但它的最主要用法还是修改被代理函数中的this引用，避免出现作用域丢失的问题。

createInterceptor(Function fcn, [Object scope])函数为当前函数设置拦截器，如下面的代码所示。

```
var sayHi = function(name){
    alert('Hi, ' + name);
}
```

```
sayHi('Fred'); // 提示"Hi, Fred"

var sayHiToFriend = sayHi.createInterceptor(function(name){
    return name == 'Brian';
});

sayHiToFriend('Fred');  // 没有提示
sayHiToFriend('Brian'); // 提示 "Hi, Brian"
```

上例中为sayHi()设置了一个拦截器，拦截器会在原函数执行之前执行，并且只有在拦截器返回true时才会去执行原函数。如果拦截器返回的值是false，就会中断执行。上例中，只有在参数为'Brian'的情况下才会返回true，所以只有参数为'Brian'时才会执行sayHi()。

createSequence(Function fcn, [Object scope])函数会使参数fcn和当前函数按顺序依次执行，如下面的代码所示。

```
var sayHi = function(name){
    alert('Hi, ' + name);
}

sayHi('Fred'); // 提示 "Hi, Fred"

var sayGoodbye = sayHi.createSequence(function(name){
    alert('Bye, ' + name);
});

sayGoodbye('Fred'); // 提示两次
```

上例中，使用了createSequence()后，会先执行sayHi()，然后执行createSequence()中设置的函数，两者依次执行。

与createInterceptor()不同的是，createInterceptor()设置的参数在原函数执行之前执行，而且createSequence()中设置的返回值也不能影响原函数的执行。

defer(Number millis, [Object obj], [Array args], [Boolean/Number appendArgs])函数会使当前函数延迟执行，如下面的代码所示。

```
var sayHi = function(name){
    alert('Hi, ' + name);
}
sayHi('Fred');

sayHi.defer(2000, this, ['Fred']);

 (function(){
    alert('Anonymous');
}).defer(100);
```

上面演示了几种使函数延迟执行的方法，如果没有使用defer()，sayHi()就会立刻执行，而defer()可以让函数延迟2秒后再执行。但是，这需要我们为sayHi()配置对应的作用域和传递参数。

11.17.4 扩展 Number

EXT只为数字类型添加了一个函数：constrain(Number min, Number max)。

我们可以使用它判断某个数字变量是否在设置的范围内。如果变量值在设置的范围内，就返回原值；否则，就返回靠近边界的数值，如下面的代码所示。

```
var num = 50;
alert(num.constrain(0,100));
alert(num.constrain(60,100));
alert(num.constrain(0,40));
```

上例中，第一次返回50，第二次返回60，第三次返回40。

11.17.5 扩展 Array

EXT中为数组对象添加了两个函数：indexOf()和remove()。

indexOf(Object o)函数首先检测传入的参数是否包含在数组中，如果o还没有加入数组，就返回-1；否则，返回o所在的索引值。remove(Object o)函数会将指定的参数从数组中删除，如果o还没有加入数组，就不会执行任何操作。

11.18 门户组件 Ext.ux.Portal

Ext.ux.Portal是以Ext.Panel为基础编写的一个扩展组件，我们可以把它当作页面上可以随意摆放的几个小窗口，如图11-23所示。

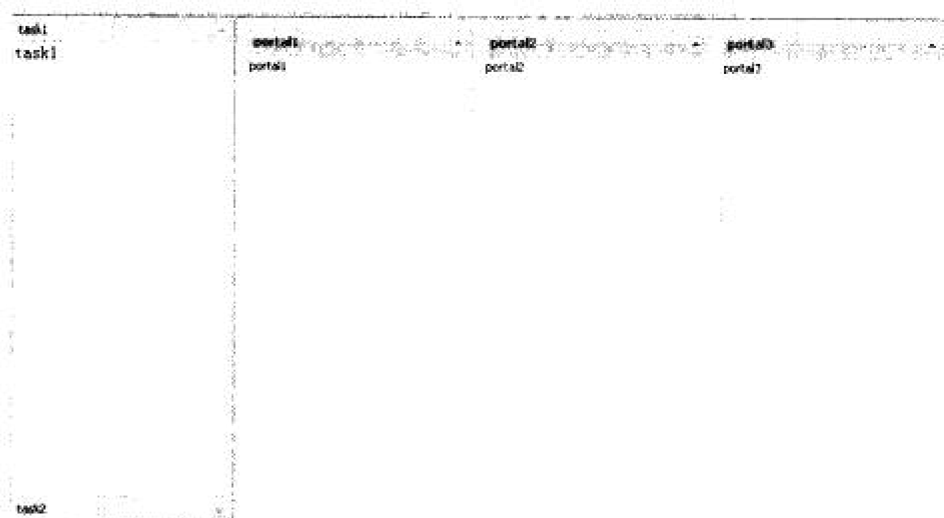


图11-23　使用Ext.ux.Portal进行页面布局

图11-23中，页面右侧部分使用的就是Ext.ux.Portal的布局方式，我们在其中摆放了3个Ext.ux.Portlet窗口，小窗口中分别显示各自的内容。可以随意拖动这3个Portlet，让它们按照我们的想法进行摆放，如图11-24所示。

实际上Ext.ux.Portal只是一个使用了ColumnLayout布局方式的Ext.Panel，在它内部使用的Ext.ux.Portal.DropZone限制了我们创建的Ext.ux.Portlet只能在本列或列与列之间进行拖动。这样，无论如何拖动，Ext.ux.Portal都可以保证最终显示的布局不会发生太大的变

化。为了保证每个Ext.ux.Portlet可以在拖动后自动适应每列的宽度，Ext.ux.Portalet使用了anchor:'100%'的参数，这样它可以在每次拖动后都以充满的方式填充到目标列中。

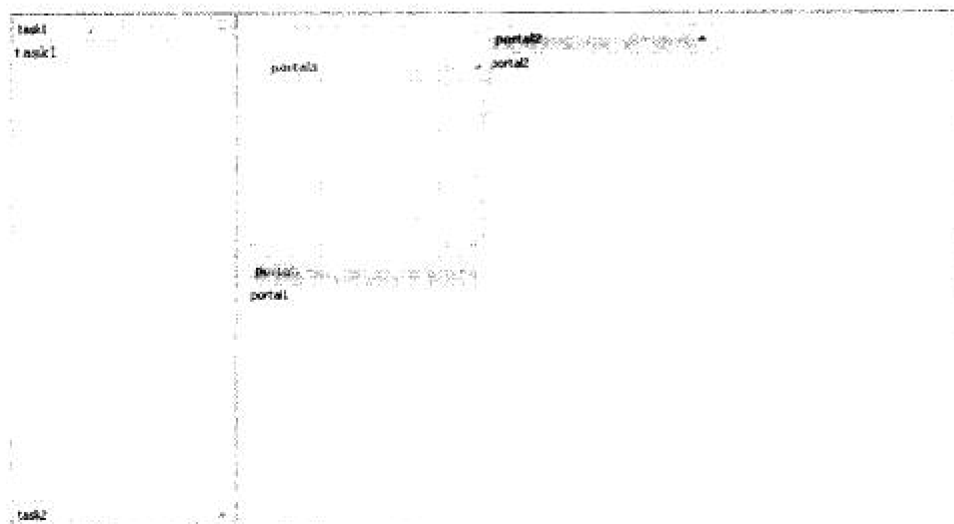


图11-24 通过拖动改变Portal布局

如果想自定义一个Ext.ux.Portal，需要首先引入examples/portal目录下的几个脚本文件和CSS样式文件，如下面的代码所示。

```
<script type="text/javascript" src="../portal/Portal.js"></script>
<script type="text/javascript" src="../portal/PortalColumn.js"></script>
<script type="text/javascript" src="../portal/Portlet.js"></script>
<link rel="stylesheet" type="text/css" href="../portal/portal.css" />
```

上面代码中引入了Portal.js、PortalColumn.js和Portlet.js这3个脚本文件，以及一个portal.css样式文件，这样就可以在项目中应用Ext.ux.Portal了。创建Ext.ux.Portal的实例的代码如下面所示：

```
var viewport = new Ext.Viewport({
    layout: 'border',
    items: [{
        region: 'west',
        width: 200,
        layout: 'accordion',
        items: [{
            title: 'task1',
            html: 'task1'
        }, {
            title: 'task2',
            html: 'task2'
        }]
    }, {
        region: 'center',
        xtype: 'portal',
        items: [{
            columnWidth: 0.33,
            style:'padding:10px 0 10px 10px',
            items:[{
                title: 'portal1',
```

```
                height: 200,
                html: 'portal1'
            }]
        }, {
            columnWidth: 0.33,
            style:'padding:10px 0 10px 10px',
            items:[{
                title: 'portal2',
                height: 200,
                html: 'portal2'
            }]
        }, {
            columnWidth: 0.33,
            style:'padding:10px 0 10px 10px',
            items:[{
                title: 'portal3',
                height: 200,
                html: 'portal3'
            }]
        }]
    }]
});
```

上例中使用了Ext.Viewport对整个页面进行布局，页面的左侧放置的是一个accordion折叠菜单，右侧部分使用的就是Ext.ux.Portal。我们将这个Portal分成三个宽度相等的列，并在每一列中都放置了一个Ext.ux.Portlet。为了便于区分，我们分别为这3个Portlet设置了标题和内容，并将它们的高度都设置为200 px，最终得到的就是图11-23中所显示的效果。

得益于EXT中灵活的布局方式，我们可以在Portlet中添加之前所讲到的任何组件，如果我们之前已经创建了表格、树形或表单，那么不需要对这些实例进行额外的配置就可以直接放入Portlet中进行显示，如下面的代码所示。

```
var viewport = new Ext.Viewport({
    layout: 'border',
    items: [{
        region: 'west',
        width: 200,
        layout: 'accordion',
        items: [{
            title: 'task1',
            html: 'task1'
        }, {
            title: 'task2',
            html: 'task2'
        }]
    }, {
        region: 'center',
        xtype: 'portal',
        items: [{
            columnWidth: 0.33,
            style:'padding:10px 0 10px 10px',
            items:[{
```

```
                title: 'portal1',
                height: 200,
                layout: 'fit',
                items: [grid]
            }]
        }, {
        columnWidth: 0.33,
            style:'padding:10px 0 10px 10px',
            items:[{
                title: 'portal2',
                height: 200,
                layout: 'fit',
                items: [tree]
            }]
        }, {
            columnWidth: 0.33,
            style:'padding:10px 0 10px 10px',
            items:[{
                title: 'portal3',
                height: 200,
                layout: 'fit',
                items: [form]
            }]
        }]
    }]
});
```

上例中，我们为3个Portlet都设置了`layout:'fit'`，然后使用`items`参数将表格、树形和表单分别放入对应的Portlet中，最终得到的效果如图11-25所示。

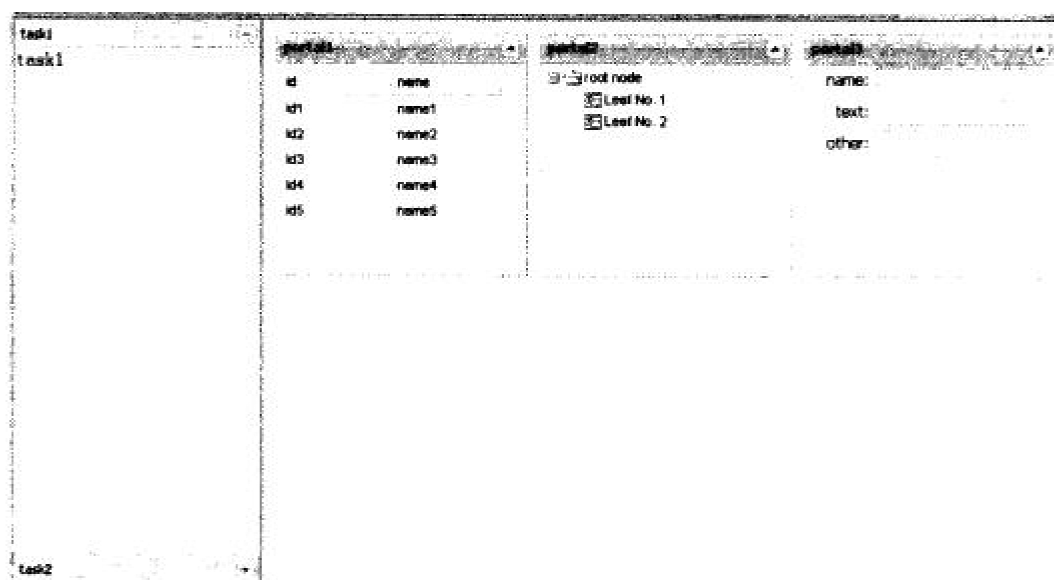


图11-25　在Portlet中使用表格、树形和表单

11.19　桌面组件 Ext.Desktop

`Ext.Desktop`是EXT提供的用于在浏览器上模拟操作系统界面的一套组件，它主要包括以下几个主要的组件。

- Ext.ux.StartMenu：模拟操作系统桌面左下方的开始菜单。
- Ext.ux.TaskBar：模拟操作系统桌面右下方的任务栏。
- Ext.Desktop：模拟整个操作系统的桌面。
- Ext.app.App：对应整个应用。
- Ext.app.Module：对应整个应用中的各个功能模块。

利用Ext.Desktop组件模拟的桌面效果如图11-26所示。

因为Ext.Desktop并不是EXT库的核心，所以在使用它之前，我们需要先将对应的脚本和CSS样式文件引入页面，如下面的代码所示。

```
<script type="text/javascript" src="../desktop/js/StartMenu.js"></script>
<script type="text/javascript" src="../desktop/js/TaskBar.js"></script>
<script type="text/javascript" src="../desktop/js/Desktop.js"></script>
<script type="text/javascript" src="../desktop/js/App.js"></script>
<script type="text/javascript" src="../desktop/js/Module.js"></script>
<link rel="stylesheet" type="text/css" href="../desktop/css/desktop.css" />
```

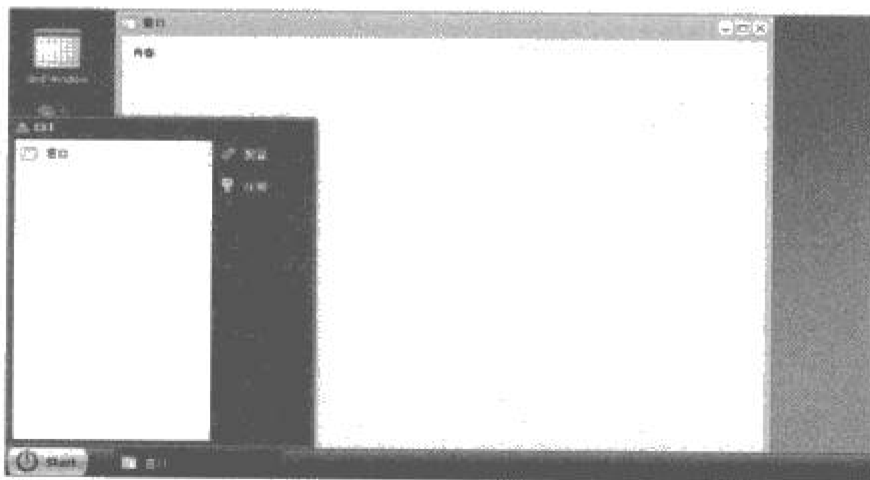


图11-26　使用Ext.Desktop组件模拟的桌面效果

我们将examples/desktop/js目录下的5个脚本文件和1个CSS样式文件引入页面，这样就可以使用Ext.Deskop模拟操作系统桌面的显示效果了。

使用Ext.Desktop的第一步是创建一个Ext.app.App的实例，如下面的代码所示。

```
MyDesktop = new Ext.app.App({
    // 初始化
    init :function(){
        Ext.QuickTips.init();
    },

    // 创建模块
    getModules : function(){
        return [
            new MyDesktop.MyModule()
        ];
    },

    // 配置开始菜单
    getStartConfig : function(){
```

```
            return {
                title: 'EXT',
                iconCls: 'user',
                toolItems: [{
                    text: '配置',
                    iconCls: 'settings',
                    scope: this
                },'-',{
                    text: '注销',
                    iconCls: 'logout',
                    scope: this
                }]
            };
        }
    });
```

与前面介绍的大多数EXT组件不同，在创建Ext.Desktop时，不需要使用Ext.onReady()指定页面加载完成后执行的初始化函数。EXT会在页面加载完成后，自动调用Ext.app.App的init()函数对整个页面进行初始化。

在init()初始化函数执行后，EXT会自动调用getModules()和getStartConfig()函数对整个应用进行配置。

getModules()函数会返回一个包含多个Ext.app.Module实例的数组，每个Ext.app.Module实例都代表应用中的一个功能模块。这些功能模块都将以弹出窗口的形式显示到桌面上，可以使用模拟桌面的开始菜单的Ext.ux.StartMenu展开这些功能模块的窗口，如图11-27所示。

对于已经展开的窗口，也可以通过模拟桌面下方的任务栏的Ext.ux.TaskBar控制某个窗口的显示或隐藏，如图11-28所示。



图11-27 在Ext.ux.StartMenu中选择展开某个功能模块的窗口



图11-28 使用Ext.ux.TaskBar隐藏窗口

Ext.app.App中的startConfig()函数主要用来配置开始菜单的选项。上例配置了两个按钮，名称分别为“配置”和“注销”，可以像之前对待普通菜单项一样对它们进行配置，设置对应的text、iconCls、scope等参数，也可以设置handler在用户点击时执行对应的操作。

在介绍完模拟桌面的整体配置之后，我们来讨论一下如何为模拟桌面创建功能模块。上例中

创建了一个名为MyDesktop.MyModel的功能模块，并在Ext.app.App的getModules()函数中对其执行了初始化操作。下面来看一下这个功能模块是如何创建的，如下面的代码所示。

```
MyDesktop.MyModule = Ext.extend(Ext.app.Module, {
    id: 'win-x',
    init : function(){
        this.launcher = {
            text: '窗口',
            iconCls: 'bogus',
            handler : this.createWindow,
            scope: this,
            windowId: 'x'
        }
    },

    createWindow : function(src){
        var desktop = this.app.getDesktop();
        var win = desktop.getWindow('window-x-win');
        if(!win){
            win = desktop.createWindow({
                id: 'window-x-win',
                title:'窗口',
                width:640,
                height:480,
                html: '<p>内容</p>',
                iconCls: 'bogus',
                shim: false,
                animCollapse: false,
                constrainHeader: true
            });
        }
        win.show();
    }
});
```

上面的示例就是我们为Ext.app.App创建的一个功能模块。在为Ext.app.App创建功能模块时，都要继承EXT提供的Ext.app.Module，这个类中只定义了一个init()函数，需要重写这个函数来实现我们的功能。

一般只需要在init()函数中定义一个launcher对象，它是一个JSON对象，内部包含了启动这个功能模块所需要的一些配置。当在Ext.ux.StartMenu中点击对应的功能模块时，就会执行launcher中定义的handler属性，弹出这个功能模块对应的窗口。

在上例中，launcher的handler属性对应着自身的createWindow()函数。在这个回调函数中，我们先通过this.app.getDesktop()获得整个应用对应的模拟桌面，然后使用desktop.getWindow('window-x-win')判断功能模块对应的窗口是否已经存在。如果窗口还没有创建，就调用desktop.createWindow()创建这个窗口，并显示出来。

除了使用Ext.ux.StartMenu显示功能窗口之外，我们还可以使用桌面上的快捷方式启动对应的功能模块，点击模拟桌面上放置的图标或链接，就可以让对应的功能窗口直接显示出来，如图11-29所示。

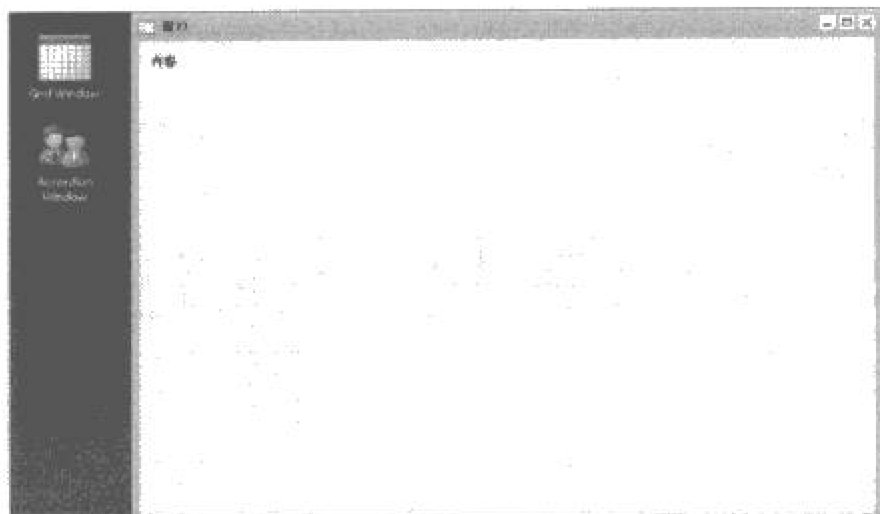


图11-29 使用快捷方式启动功能模块

Ext.Desktop中将快捷方式称为shortcut，我们不需要写任何代码来配置快捷方式，只需要在为标签和对象命名时遵守一定的规则即可。

在模拟桌面上显示的快捷方式时所使用的HTML标签如下所示。

```
<dl id="x-shortcuts">
  <dt id="win-x-shortcut" class="grid">
    <a href="#"><img src="../desktop/images/s.gif" />
    <div>Grid Window</div></a>
  </dt>
  <dt id="win-x-shortcut" class="im">
    <a href="#"><img src="../desktop/images/s.gif" />
    <div>Accordion Window</div></a>
  </dt>
</dl>
```

如上面的代码所示，模拟桌面上的快捷方式都必须包含在id="x-shortcuts"的dl标签中，dl标签中包含的每个dt标签都将成为一个快捷方式。这些dt标签的id属性都以-shortcut结尾，将id属性中的-shortcut部分去掉后，得到的就应该是这个快捷方式所对应的功能模块的id。比如，上例中<dt id="win-x-shortcut">对应的功能模块就是id:'win-x'的功能模块。

至此，我们将Ext.Desktop模拟桌面组件完整地介绍了一遍，希望大家可以使用它制作出自己的模拟桌面。

11.20 小结

本章首先介绍了EXT提供的各种工具函数，然后介绍了如何使用DomHelper和Template生成DOM片段，重点介绍了模板相关的Template类和XTemplate类。同时还讨论了如何为EXT中的组件设置提示信息，并介绍了多种配置方法。

本章演示了如何在项目中使用EXT提供的悬停提示，并提供了多种配置方式。此外还讨论了如何使用Ext.state保存组件的状态及可能引发的问题，并给出了解决方案。而且介绍了EXT中包含的各种辅助工具类和EXT对JavaScript的一些扩展。

最后，本章展示了Ext.ux.portal和Ext.Desktop这两个精美的组件。

第12章 一个完整的EXT应用

本章内容

- 确定整体布局
- 使用HTML和CSS设置静态信息
- 对学生信息进行数据建模
- 在页面中显示学生信息列表
- 添加表单编辑学生信息
- 为表单添加提交事件
- 清空表单信息
- 删除指定的学生信息
- 在表格和表单之间进行数据交互
- 提升加载速度

本章将综合运用前面所学的知识，开发一个简单的学生信息管理系统（如图12-1所示）。该系统的主要功能包括：显示学生信息、添加学生信息、修改学生信息，以及删除学生信息。这些功能的实现非常简单，这里将演示如何在EXT中实现这些常用功能。

图12-1　学生信息管理系统界面

12.1 确定整体布局

在动手实现这些功能操作之前，首先应该确定页面的整体布局。在这里，我们用BorderLayout把页面分隔成4个部分：最上方显示系统的名称，最下方显示版权信息，中间部分左侧显示学生信息列表，中间部分右侧中的表单用来添加或修改学生信息。

实现上述布局效果的代码如代码清单12-1所示。

代码清单12-1 实现学生信息管理系统的布局

```
Ext.onReady(function() {
    var viewport = new Ext.Viewport({
        layout: 'border',
        items: [{
            region: 'north',
            html: 'head'
        }, {
            region: 'center',
            html: 'grid'
        }, {
            region: 'east',
            html: 'form'
        }, {
            region: 'south',
            html: 'foot'
        }]
    });
});
```

使用Ext.Viewport为整个页面进行布局设置，其中每个部分都直接用HTML参数做了标记。这个布局的显示效果如图12-2所示。

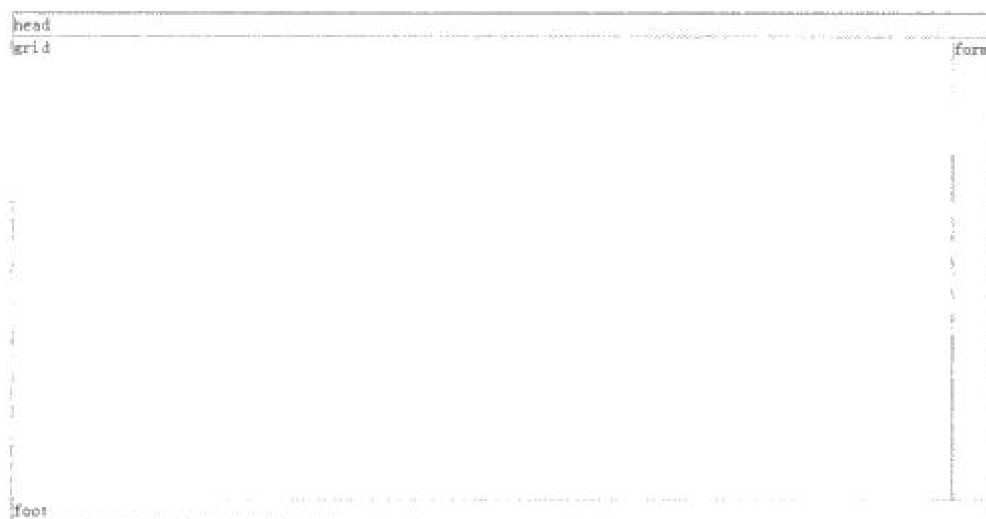


图12-2 学生信息管理系统布局效果图

如图12-2所示，现在看到的只是一个空白的框架，每一部分的具体内容都需要进一步去实现。无论该学生信息管理系统的功能多么复杂，都不会超出目前框架中的布局设计。接下来，我们来逐一实现各个部分的功能。

12.2　使用 HTML 和 CSS 设置静态信息

用于显示标题和版权信息的文字都是静态的，可以直接调用HTML中设置好的内容。在Ext.Panel中添加静态信息的方式有如下两种。

- HTML参数：它让我们可以直接在JavaScript脚本中写上静态信息的内容。
- contentEl参数：它引用页面中某一个div的id，在显示Panel时将这个div中的内容显示在对应的布局域中。

相对而言，HTML参数更适合简单的内容。如果需要引入复杂格式的静态信息，还是应该使用contentEl参数。在这个示例中，我们就选择了contentEl参数为头部和底部制定静态信息部分的内容，实现方法如代码清单12-2所示。

代码清单12-2　在布局中设置静态内容

```
var viewport = new Ext.Viewport({
    layout: 'border',
    items: [{
        region: 'north',
        contentEl: 'head'
    }, {
        region: 'center',
        html: 'grid'
    }, {
        region: 'east',
        html: 'form'
    }, {
        region: 'south',
        contentEl: 'foot'
    }]
});
```

在上面的代码中，显示在上方的north部分引用的contentEl是'head'，它在页面中的内容如下所示。

```
<div id="head" style="font-weight:bold;font-size:200%;">学生信息管理</div>
```

显示在下方的south部分引用的contentEl是'foot'，它在页面中的内容如下所示。

```
<div id="foot" style="text-align:right;"> - &copy; 2008 <a
href="http://www.family168.com" target="_blank">www.family168.com</a> - </div>
```

因为这两部分的标签内容都会在EXT进行页面布局时重新提取和设置，所以在开始时不用考虑把这两个div写到页面的什么位置，只要把它们写到页面里，EXT就会自动进行布局，把它们放到预设的位置。

设置过contentEl后，整个页面就变成了图12-3的样子。这里演示的效果比较简单，结合使用了HTML标签和CSS，为标题设置字体（加粗）和字号（变大），版权信息则是右对齐并设置了超链接。在实际项目中，可以把美工设计出来的页面标签直接复制到对应的div下，刷新页面后就会显示在对应的位置。



图12-3 上下部分加入静态信息的效果

12.3 对学生信息进行数据建模

接下来我们要实现对学生信息进行实际操作的功能。用Java编写后台脚本，用Hsqldb数据库作为保存数据的介质。首先要在数据库中创建学生信息数据表，如代码清单12-3所示。

代码清单12-3 创建学生信息表

```
create table student(
    id bigint, -- 主键
    code varchar(50), -- 学号
    name varchar(50), -- 姓名
    sex integer, -- 性别
    age integer, -- 年龄
    political varchar(50), -- 政治面貌
    origin varchar(50), -- 籍贯
    professional varchar(50) -- 所属系
);
alter table student
    add constraint pk_student primary key (id);
alter table student
    alter column id bigint generated by default as identity (start with 1,increment by 1);
```

这张student表中包含8个字段，分别对应学生的各种详细信息。最后两行不是标准的ANSI SQL语句，这是Hsqldb中的特有功能，表示把id字段设置成student表的唯一主键，并由数据库服务器控制自动增长。

在本示例中将它作为嵌入式内存数据库，只要把hsqldb-1.8.0_7.jar放到WEB-INF/lib目录下就可以了，不必再去安装和配置外部服务器。Hsqldb数据库会在JDBC第一次连接时启动，从WEB-INF/classes目录下的test.scripts和test.properties两个文件中加载初始数据，其后的所有操作都会在内存中进行，唯一的缺陷是以这种方式运行的数据都保存在内存里。一旦服务器关闭就会导致数据丢失，下次重新启动数据库时将无法得知上次执行过何种操作，我们看到的数据依然是从test.scripts和test.properties中读取的初始化数据。有关Hsqldb数据库的详细信息可以去它的官方网站www.hsqldb.org查看。

在下面的讨论中，我们将尽量使用标准的ANSI SQL语句，保证可以在不同的数据库上正常

运行。对于不得不使用Hsqldb数据库的特定功能的SQL语句，我们会对它们进行单独讨论。

对应已经建立好的数据表结构，我们编写了一个与数据库表结构对应的JavaBean——Student.java，这个JavaBean中的字段与数据库表中的字段是一一对应的，如代码清单12-4所示。

代码清单12-4 学生信息领域模型

```
package com.family168.student;

public class Student {
    private long id;
    private String code;
    private String name;
    private int sex;
    private int age;
    private String political;
    private String origin;
    private String professional;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
}
// 获取方法和设置方法
}
```

这是一个简单的JavaBean，它的每个属性对应了student表中的一个字段，我们将数据库中的字段映射为Java对象，在后面的操作里就可以为它们添加特定的逻辑操作。

直接操作Student.java的类叫做StudentDao.java，DAO是Data Access Object的简写，它主要负责Student.java和数据库之间的数据转换。比如，pagedQuery()会把从数据库中读取的数据放入Student.java对象中，并按照特定的形式返回。insert()和update()方法则是将Student.java中的数据保存到数据库中。remove()方法执行的是删除操作，它会根据指定的id从数据库中删除对应的对象。

这些对数据库的操作实际上都是大同小异的，每次先打开与数据库的连接，然后执行查询或更新操作，最后关闭连接并释放资源。

我们将获得数据库连接的部分代码封装在一个叫做DbUtils.java的工具类中，如代码清单12-5所示。

代码清单12-5 数据库工具类

```
package com.family168.student;

import java.sql.*;

public class DbUtils {
    static {
        try {
```

```
            Class.forName("org.hsqldb.jdbcDriver");
        } catch(Exception ex) {
            System.err.println(ex);
        }
    }

    static Connection getConn() throws Exception {
        return DriverManager.getConnection("jdbc:hsqldb:res:/test", "sa", "");
    }

    static void close(ResultSet rs, Statement state, Connection conn) {
        if (rs != null) {
            try {
                rs.close();
            } catch(SQLException ex) {
                ex.printStackTrace();
            }
            rs = null;
        }
        if (state != null) {
            try {
                state.close();
            } catch(SQLException ex) {
                ex.printStackTrace();
            }
            state = null;
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (Exception ex) {
                ex.printStackTrace();
            }
            conn = null;
        }
    }
}
```

这是一个工具类，不需要使用new关键字实例化就可以直接调用其中的方法，static{}静态初始化时加载Hsqldb的JDBC驱动，其后每次执行getConn()方法就可以得到一个与数据库的连接。close()方法提供了一种关闭数据库连接并释放资源的简便方法，使用它可以一次性关闭ResultSet、Statement和Connection 3个对象，方法内自动检测对象是否为null，可以放心使用。

在StudentDao.java中统一通过DbUtils.java来获得数据库连接，以此实现对数据库的操作，比如remove()方法中就是先通过DbUtils的getConn()方法获得数据库的连接再进行删除操作的，如下面的代码所示。

```
public void remove(long id) throws Exception {
    String sql = "delete from student where id=?";

    Connection conn = DbUtils.getConn();
    PreparedStatement state = conn.prepareStatement(sql);
```

```
        state.setLong(1, id);

        state.executeUpdate();
        DbUtils.close(null, state, conn);
    }
```

先用DbUtils.getConn()获得数据库链接，然后准备SQL语句对应的Statement。设置指定的id后，调用Statement和executeUpdate()执行删除操作，最后不要忘记用DbUtils.close()关闭与数据库的连接释放资源。

另外两种更新数据库的方法与remove()相似，insert()和update()分别使用的是SQL中的insert和update语句，再将作为参数的Student.java对象设置到Statement中执行更新，实现代码请参考StudentDao.java中的内容，这里就不再赘述了。

pagedQuery()中的查询功能比较复杂，需要详细讨论一下。下面看一下其中的代码部分，如代码清单12-6所示。

代码清单12-6　pagedQuery()

```
public Page pagedQuery(int start, int limit, String sort, String dir) throws
Exception {

    String sql = "select limit " + start + " " + limit + " * from student";
    if (sort != null && !sort.equals("") && dir != null && !dir.equals("")) {
        sql += " order by " + sort + " " + dir;
    }
    Connection conn = DbUtils.getConn();
    Statement state = conn.createStatement();

    ResultSet rs = state.executeQuery(sql);
    List result = new ArrayList();
    while (rs.next()) {
        Student student = new Student();
        student.setId(rs.getLong("id"));
        student.setCode(rs.getString("code"));
        student.setName(rs.getString("name"));
        student.setSex(rs.getInt("sex"));
        student.setAge(rs.getInt("age"));
        student.setPolitical(rs.getString("political"));
        student.setOrigin(rs.getString("origin"));
        student.setProfessional(rs.getString("professional"));
        result.add(student);
    }
    rs = state.executeQuery("select count(*) from student");
    int totalCount = 0;
    if (rs.next()) {
        totalCount = rs.getInt(1);
    }
    DbUtils.close(rs, state, conn);

    Page page = new Page(totalCount, result);
    return page;

}
```

pagedQuery()方法的4个参数分别是start、limit、sort和dir。start和limit用来进行分页查询，start表示从第几条数据进行查询，limit表示最多返回几条查询数据。sort和dir用来对查询的结果进行排序，sort表示对哪个字段进行排序，dir表示排序时使用升序还是降序。

为了实现分页和排序功能，pagedQuery()方法首先要根据传递过来的参数生成对应功能的SQL语句。

```
String sql = "select limit " + start + " " + limit + " * from student";
```

上面是从student表中进行查询，并依据start和limit的内容进行分页。这里的"select limit " + start + " " + limit不是标准的ANSI SQL语句，而是Hsqldb专门为分页查询提供的功能。很遗憾的是，标准ANSI SQL中没有提供分页查询的功能，基本上每个主流数据库都提供了自己的分页查询方式，这些方式又完全不相同。如果你想把这个示例转换到其他数据库上，必须将这里的分页查询部分修改为对应数据库的SQL语句。

```
if (sort != null && !sort.equals("") && dir != null && !dir.equals("")) {
sql += " order by " + sort + " " + dir;
}
```

生成排序功能的SQL语句比较简单，只需要判断sort和dir是否为空。如果不为空，就可以直接附加到原来的SQL的后面，对获得的查询结果实现排序。

得到了需要的SQL语句之后，我们立刻进行查询，通过Statement和ResultSet把每一条返回的记录都转换成Student.java对象，放入到ArrayList中。

接下来还需要获取数据库中的总记录数，对应的SQL语句为select count(*) from student，这是一条标准的ANSI SQL语句，不用做任何修改就可以在所有的主流数据库上运行。使用这条SQL语句，我们获得了数据库中保存的学生信息的总数。

现在可以关闭数据库连接，把学生信息总数totalCount和本页显示的学生信息列表result放入Page.java对象中并返回。

至此，我们完成了数据建模部分代码的编写，对应的源代码放在WEB-INF/src目录下，编译后的代码放在WEB-INF/classes目录下。供对应的JSP或其他Java类调用，通过这些类，我们可以访问数据库获得需要的查询结果，也可以通过这些类对数据库中的数据进行更新。

12.4 在页面中显示学生信息列表

后台的数据库和Java代码都已经准备妥当，现在编写显示学生信息列表的表格部分的代码，如代码清单12-7所示。

代码清单12-7 前台表格部分的实现代码

```
var sexRenderer = function(value) {
    if (value == 1) {
        return '<span style="color:red;font-weight:bold;">男</span>';
    } else if (value == 2) {
        return '<span style="color:green;font-weight:bold;">女</span>';
    }
};
```

```
var StudentRecord = Ext.data.Record.create([
    {name: 'id', type: 'int'},
    {name: 'code', type: 'string'},
    {name: 'name', type: 'string'},
    {name: 'sex', type: 'int'},
    {name: 'age', type: 'int'},
    {name: 'political', type: 'string'},
    {name: 'origin', type: 'string'},
    {name: 'professional', type: 'string'}
]);

var store = new Ext.data.Store({
    proxy: new Ext.data.HttpProxy({url: './jsp/list.jsp'}),
    reader: new Ext.data.JsonReader({
        totalProperty: 'totalCount',
        root: 'result'
    },StudentRecord),
    remoteSort: true
});
store.load({params:{start:0,limit:15}});

var columns = new Ext.grid.ColumnModel([
    {header: '学号', dataIndex: 'code'},
    {header: '姓名', dataIndex: 'name'},
    {header: '性别', dataIndex: 'sex', renderer: sexRenderer},
    {header: '年龄', dataIndex: 'age'},
    {header: '政治面貌', dataIndex: 'political'},
    {header: '籍贯', dataIndex: 'origin'},
    {header: '所属系', dataIndex: 'professional'}
]);
columns.defaultSortable = true;

var grid = new Ext.grid.GridPanel({
    title: '学生信息列表',
    region: 'center',
    loadMask: true,
    store: store,
    cm: columns,
    sm: new Ext.grid.RowSelectionModel({singleSelect:true}),
    viewConfig: {
        forceFit: true
    },
    bbar: new Ext.PagingToolbar({
        pageSize: 15,
        store: store,
        displayInfo: true
    })
});
```

sexRenderer是一个工具函数，它用来在表格中显示学生的性别。数据库中sex字段的类型为int，我们用1代表“男”，2代表“女”，sexRenderer中使用if语句判断当前行的sex值。在1的情况下显示红色粗体的“男”，在2的情况下显示绿色粗体的“女”。之后sexRenderer会作为

ColumnModel的一部分设置到表格中，负责显示“性别”这一列的内容。

StudentRecord部分使用Ext.data.Record的create()函数创建了一个类，就像之前在数据建模的过程中使用JavaBean对数据库进行映射一样，EXT中也对student的数据进行了封装。这样就可以在表格的store中和表单的reader中直接使用这个预定义的类型，避免了重复定义相同的数据类型。

接下来的Ext.data.Store就利用了上面的StudentRecord类型，处理从后台获得的信息。它使用Ext.data.HttpProxy从jsp/list.jsp中获得学生信息列表的信息，返回信息中的totalProperty和root两个参数分别指定了后台数据的记录总数和当前页面显示信息的队列，这些数据最终都会显示在表格中。

创建好store之后，随即调用store.load({params:{start:0,limit:15}});进行分页查询，这里传递的两个参数start和limit，表示从第一条记录开始查询，最多获得15条记录。这部分与后台的jsp/list.jsp交互的操作将在后面详细讨论。

下面的工作是创建Ext.grid.ColumnModel，将表格中每列显示的数据与store中的数据相对应。建立好列模型后，再调用columns.defaultSortable = true;将所有列都设置成可排序的。排序功能也可以通过为每一列设置sortable:true来实现，但是逐一设置会比较烦琐，不如统一设置方便。

万事俱备只欠东风，表格需要的组件部分都准备好了，现在可以创建表格来显示学生信息列表了。为了不使表格显得寒酸，除了上面提到的store和columns之外，我们还为表格设置了标题title。用loadMask:true开启读取数据时的提示功能，这会在每次store去后台读取数据时自动显示等待提示信息。sm: new Ext.grid.RowSelectionModel({singleSelect:true})限制用户每次只能选中一行，这是为了后面与表单进行互操作所做的准备。viewConfig:{forceFit: true}自动调整每列的宽度，使整个表格更加饱满。最后还为bbar添加了分页工具条，可以用它进行表格的分页跳转和数据刷新操作。

Region:'center'表示把这个表格放到BorderLayout的中间位置。接下来，我们调整原来Viewport中的配置，把之前放在中间的Panel替换为创建好的表格，如代码清单12-8所示。

代码清单12-8 将表格加入页面

```
var viewport = new Ext.Viewport({
    layout: 'border',
    items: [{
        region: 'north',
        contentEl: 'head'
    }, grid, {
        region: 'east',
        html: 'form'
    }, {
        region: 'south',
        contentEl: 'foot'
    }]
});
```

最终的显示效果如图12-4所示。

12

学生信息管理

学生信息列表　　form

学号	姓名	性别	年龄	政治面貌	籍贯	所属系
2002015	[illegible]	男	21	团员	湖北省	物流工程学院
2002002	[illegible]	男	22	党员	河南省	动力工程系
2002003	魏心	男	23	团员	四川省	管理学院
2002004	王小[illegible]	男	23	团员	重庆市	材料学院
2002005	[illegible]	男	22	党员	河北省	文法学院系
2002006	[illegible]	男	23	群众	香港特别行政区	计算机学院
2002007	[illegible]	女	22	团员	山西省	计算机学院
2002008	[illegible]	女	22	党员	北京市	管理学院
2002009	[illegible]	女	23	团员	安徽省	机械学院
2002010	[illegible]	女	20	团员	广西	文法学院
2002011	[illegible]	男	25	党员	安徽省	体育系
2002012	[illegible]	男	26	团员	江苏省	机械学院
2002016	[illegible]	男	24	团员	黑龙江省	环境工程系
2002014	[illegible]	女	21	党员	福建省	外语系
2002001	[illegible]	女	20	群众	浙江省	计算机学院

第 1 页共 2 页　　显示1 - 15，共16条

- © 2008 www.family168.com -

图12-4　加入表格后的页面效果

因为我们没有为右侧的表单部分设置宽度，所以Viewport自动为它计算了一个最小宽度，结果位于中间的表格几乎布满了整个页面。从图12-4中可以看到，表格中显示了每列对应的数据，“性别”这一列也按照sexRenderer中定义的那样显示得错落有致。现在可以单击表格下部的分页工具条上的按钮查看分页查询的效果，也可以单击某一列的首部，查看按列排序的功能。

看过了页面的效果，我们再回到后台看看为前台提供数据的list.jsp中的内容，如代码清单12-9所示。

代码清单12-9　list.jsp

```
<%@ page contentType="application/json;charset=utf-8" import="com.family168.student.
*" %><%
    request.setCharacterEncoding("utf-8");
    response.setCharacterEncoding("utf-8");
    int start = 0;
    try {
        start = Integer.parseInt(request.getParameter("start"));
    } catch(Exception ex) {
        System.err.println(ex);
    }
    int limit = 15;
    try {
        limit = Integer.parseInt(request.getParameter("limit"));
    } catch(Exception ex) {
        System.err.println(ex);
    }
    String sort = request.getParameter("sort");
    String dir = request.getParameter("dir");

    StudentDao dao = StudentDao.getInstance();
    Page pager = dao.pagedQuery(start, limit, sort, dir);

    out.print(pager.toString());
%>
```

list.jsp中的代码可以分成3部分，如下所述。

(1) 第一部分，设置JSP使用的contentType和encoding，因为Ajax在访问后台时，默认使用UTF-8作为请求和响应时传递的数据的默认编码，而在Tomcat服务器中，使用的默认编码是ISO-8859-1。这种编码的差异会在传递中文字符时出现乱码，所以首先需要把请求和响应的编码都统一设为UTF-8。

(2) 第二部分，处理前台传递的参数，包括start、limit、sort、dir等4个参数，这4个参数在12.3节中已经讨论过。start和limit用来进行分页查询，start表示从第几条数据开始查询，limit表示最多返回几条查询数据。sort和dir用来对查询的结果进行排序，sort表示对哪个字段排序，dir表示排序时使用升序还是降序。

因为使用HTTP协议只能传递字符类型的参数，所以在list.jsp中，我们要对这几个参数进行类型转换。注意，在有可能出现转换失败的情况时，需要为对应的参数设置默认值，这里默认设置start为0，limit为15。

(3) 第三部分，将获得的4个参数传递给StudentDao，获得分页结果对象，最后调用Page的toString()方法，将它生成的内容返还给前台处理。

将StudentDao设计成一个单例模式（singleton），这样可以在当前系统中保证只创建一个实例，这个实例被其他类所共享，避免了重复创建对象造成的资源浪费。

Page的toString()方法将分页结果转换成JSON格式，这个功能是通过Student.java和Page.java两个类中的toString()方法来实现的。

Student.java的代码如下所示：

```
public String toString() {
    return "{id:" + id +
        ",code:'" + code +
        "',name:'" + name +
        "',sex:" + sex +
        ",age:" + age +
        ",political:'" + political +
        "',origin:'" + origin +
        "',professional:'" + professional +
        "'}";
}
```

Page.java的代码如下所示：

```
public String toString() {
    return "{totalCount:" + totalCount + ",result:" + result + "}";
}
```

通过这两个方法，page.toString()会返回一个符合JSON格式的字符串，并使用之前设置好的UTF-8编码格式发送给前台。前台EXT接收到的内容如代码清单12-10所示。

代码清单12-10　前台EXT获得的JSON数据

```
{
totalCount:16,
```

```
result:[{

id:1,
code:'2002015',
name:'张光和',
sex:1,
age:21,
political:'团员',
origin:'湖北省',
professional:'物流工程学院'
}, {
id:2,
code:'2002002',
name:'张值强',
sex:1,
age:22,
political:'党员',
origin:'河南省',
professional:'动力工程系'
}]
}
```

totalCount表示数据库中现有学生信息的总数，result表示当前页显示的信息列表，前台获得这些信息之后，就可以把这些数据显示到表格中。

12.5 添加表单编辑学生信息

配置好表格之后，添加一个表单来编辑学生信息，如代码清单12-11所示。

代码清单12-11 编辑学生信息的表单

```
var form = new Ext.form.FormPanel({
    title: '编辑学生信息',
    region: 'east',
    frame: true,
    width: 300,
    autoHeight: true,
    labelAlign: 'right',
    labelWidth: 60,
    defaultType: 'textfield',
    defaults: {
        width: 200,
        allowBlank: false
    },
    items: [{
        xtype: 'hidden',
        name: 'id'
    },{
        fieldLabel: '学号',
        name: 'code'
    },{
        fieldLabel: '姓名',
        name: 'name'
```

```
        },{
            fieldLabel: '年龄',
            name: 'age',
            xtype: 'numberfield',
            allowNegative: false
        },{
            fieldLabel: '性别',
            name: 'sexText',
            hiddenName: 'sex',
            xtype: 'combo',
            store: new Ext.data.SimpleStore({
                fields: ['value','text'],
                data: [['1','男'],['2','女']]
            }),
            emptyText: '请选择',
            mode: 'local',
            triggerAction: 'all',
            valueField: 'value',
            displayField: 'text',
            readOnly: true
        },{
            fieldLabel: '政治面貌',
            name: 'political',
            xtype: 'combo',
            store: new Ext.data.SimpleStore({
                fields: ['text'],
                data: [['群众'],['党员'],['团员']]
            }),
            emptyText: '请选择',
            mode: 'local',
            triggerAction: 'all',
            valueField: 'text',
            displayField: 'text',
            readOnly: true
        },{
            fieldLabel: '籍贯',
            name: 'origin'
        },{
            fieldLabel: '所属系',
            name: 'professional'
        }],
        buttons: [{
            text: '添加'
        },{
            text: '清空'
        },{
            text: '删除'
        }]
    });
```

我们把这个表单的宽度设置为300，让它里面的输入组件对应的文字标签都右对齐，其中的defaults参数很有用，在它里面设置的参数会自动赋予表单内部的输入组件，这样内部组件的相同配置只需要配置一次就可以了。我们使用的配置是{ width: 200, allowBlank: false},

每个组件的宽度都设置为80，而且不能提交空值。因为我们使用的输入组件大多是textfield，所以把defaultType设置为'textfield'可以不必为每个items设置xtype。

在对Ext.form.FormPanel进行了这些设置之后，我们可以把学生信息对应的输入控件放到items中。其中“学号”，“姓名”，“籍贯”，“所属系”都是textfield类型，因此只需要配置name和fieldLabel。

需要使用xtype:'hidden'将id字段配置为隐藏域。虽然在页面上看不到它，但是也可以通过后面的loadRecord()方法设置id字段的数据。在提交表单时，它里边的数据也会一起发送到后台，它的作用和<input type="hidden" name="id">是一致的。

“年龄”输入框限制用户只能输入数字，使用了Ext.form.NumberField数字输入组件，对应使用了xtype:'numberfield'。因为人的年龄不可能是负数，所以我们还将allowNegative设置为false，不允许用户输入负数。

虽然“性别”和“政治面貌”使用的都是Ext.form.ComboBox，但还是有一些不同。

虽然“性别”这个ComboBox选择的是“男”和“女”，但是实际上传递到后台的是对应的1或2两个整数。因此，在配置ComboBox时需要将displayField和valueField分开使用，还要配置上hiddenName，否则发送到后台的依然是显示在页面上的“男”和“女”，而不是与之对应的1或2两个整数。

“政治面貌”这个ComboBox就简单了许多，选择的文字和传递给后台的数据是一致的，只需displayField和valueField的配置相同，不需要额外配置hiddenName参数。

经过了这一系列的配置后，我们得到了一个可以用来编辑学生信息的表单，效果如图12-5所示。

到此为止，学生信息管理系统的整个界面已经制作完成，但是还不能利用表单对学生信息执行添加、修改和删除等操作。

图12-5　加入表单后的界面效果

下面要为表单的按钮设置监听事件，在单击相应按钮时执行对应的操作。

12.6 为表单添加提交事件

在上面的表单中，我们放置了3个按钮，分别是“添加”、“清空”和“删除”。首先我们为“添加”按钮添加事件，为它的handler参数指定一个处理函数，如代码清单12-12所示。

代码清单12-12 表单按钮的单击事件

```
{
        text: '添加',
        handler: function() {
            if (!form.getForm().isValid()) {
                return;
            }
            if (form.getForm().findField("id").getValue() == "") {
                // 添加
                form.getForm().submit({
                    url: './jsp/save.jsp',
                    success: function(f, action) {
                        if (action.result.success) {
                            Ext.Msg.alert('消息', action.result.msg,function(){
                                grid.getStore().reload();
                                form.getForm().reset();
                                form.buttons[0].setText('添加');
                            });
                        }
                    },
                    failure: function() {
                        Ext.Msg.alert('错误', "添加失败");
                    }
                });
            } else {
                // 修改
                form.getForm().submit({
                    url: './jsp/save.jsp',
                    success: function(f, action) {
                        if (action.result.success) {

                            Ext.Msg.alert('消息',action.result.msg,function(){

                                grid.getStore().reload();
                                form.getForm().reset();
                                form.buttons[0].setText('添加');
                            });
                        }
                    },
                    failure: function() {
                        Ext.Msg.alert('错误', "修改失败");
                    }
                });
            }
        }
    }
```

12

在handler处理函数中，首先调用form.getForm().isValid()进行数据校验。如果返回false，说明表单中某些输入组件里的数据还无法通过校验，不应该提交这些错误格式的数据，这时我们应该直接跳出函数，中止提交操作。

如果表单顺利通过数据校验检测，我们便进入下一步，准备向后台提交数据。还记得我们刚才讲过的id对应的隐藏域吗？这里就是通过它的数值来判断本次提交是添加操作还是修改操作。如果form.getForm().findField("id").getValue()为空字符串，就是在添加数据；如果它不为空字符串，就是在修改一个已有的数据。

虽然我们对添加和修改操作进行了区分，但是在实际提交代码时并没有太大区别，只是在提交错误时分别提示“添加失败”和“修改失败”而已。让我们通过代码清单12-13来看看这些数据是如何提交给后台的。

代码清单12-13　将数据提交给后台

```
// 添加
form.getForm().submit({
url: './jsp/save.jsp',
success: function(f, action) {
    if (action.result.success) {
        Ext.Msg.alert('消息', action.result.msg, function() {
            grid.getStore().reload();
            form.getForm().reset();
            form.buttons[0].setText('添加');
        });
    }
},
failure: function() {
Ext.Msg.alert('错误', "添加失败");
}
});
```

这里的form表示我们前面创建的Ext.form.FormPanel，它的getForm()函数返回FormPanel内部对应的Ext.form.BasicForm。现在我们调用BasicForm的submit()函数，将内部items中输入组件的值提交给后台的jsp/save.jsp。

如果后台没有出现异常，而且返回的JSON信息中包含{success:true}，那么就会执行success参数对应的处理函数。在success处理函数中，我们创建一个Ext.Msg.alert()显示响应的JSON信息中的msg部分的内容。在用户关闭alert提示框之后，调用grid.getStore().reload()刷新表格中的数据，同时调用form.getForm().reset()清空上次提交的数据。

如果后台出现400或500错误，就会触发failure参数对应的处理函数。这里只是弹出一个alert提示框告诉用户“添加失败”，等待用户对刚才提交失败的信息进行修改或做其他处理。

在添加和修改信息时，都是提交给后台的jsp/save.jsp进行处理。save.jsp这个文件同时负责添加和修改两个操作，它里面的内容如代码清单12-14所示。

代码清单12-14　save.jsp

```
<%@ page contentType="application/json;charset=utf-8" import="com.family168.student.
```

```
*" %><%
    request.setCharacterEncoding("utf-8");
    response.setCharacterEncoding("utf-8");

    String id = request.getParameter("id");
    String code = request.getParameter("code");
    String name = request.getParameter("name");
    String sex = request.getParameter("sex");
    String age = request.getParameter("age");
    String political = request.getParameter("political");
    String origin = request.getParameter("origin");
    String professional = request.getParameter("professional");

    Student student = new Student();
    student.setCode(code);
    student.setName(name);
    student.setSex(Integer.parseInt(sex));
    student.setAge(Integer.parseInt(age));
    student.setPolitical(political);
    student.setOrigin(origin);
    student.setProfessional(professional);

    StudentDao dao = StudentDao.getInstance();
    if (id == null || id.equals("")) {
        dao.insert(student);
    } else {
        student.setId(Long.parseLong(id));
        dao.update(student);
    }
    out.print("{success:true,msg:'保存成功'}");
%>
```

与之前讨论过的list.jsp类似，它里面的处理过程也大致分为 3 步。

(1) 设置JSP使用的contentType和encoding，把请求和响应的编码都统一为UTF-8。

(2) 处理前台传递的参数，通过request.getParameter()方法从请求中获得刚刚提交过来的学生信息，创建一个Student.java对象，将对应的学生信息添加到对象中。这个过程中要注意对age和sex两个参数进行数据类型转换，因为从HTTP请求中只能获得字符串，而它们都需要在其后转换为整数类型。

为了在后面的操作中区分添加和修改操作，我们没有对参数id进行处理。

(3) 现在我们已经获得了提交数据，并把这些数据都放到了对应的Student.java对象中。现在我们可以判断参数id的值，以此来判断后面要执行的是添加操作还是修改操作。

与前台EXT提交数据代码相似，如果id == null || id.equals("")，就表明应该执行添加操作，执行StudentDao的insert()方法。否则，应该执行修改操作，执行StudentDao的update()方法。

最后，只要未出现异常，我们就认为添加或修改操作成功了，直接向response中写入提示成功信息的JSON字符串。

```
out.print("{success:true,msg:'保存成功'}");
```

12

12.7　清空表单信息

再来看看“清空”按钮中handler对应的处理函数。

```
{
    text: '清空',
    handler: function() {
        form.getForm().reset();
        form.buttons[0].setText('添加');
    }
}
```

它的作用是调用form.getForm().reset();清空表单中的所有数据，然后把form.buttons[0]也就是表单的第一个按钮的文字修改为“添加”。

reset将表单恢复到初始状态。如果刚才在表单中添加了一些不必要的数据，又不想逐一去删除它们，那么只需要单击这个按钮就能清除所有输入框中的数据了。

Reset还有一个功能是清空隐藏域id中的数据，因为id在页面上是看不到的，所以无法手工删除它的数据，此时就只好求助于“清空”按钮了。如果想从“修改”状态转换到“添加”状态，也需要它的帮助。

form.buttons[0]从表单中取出排在第一位的按钮。我们一共设置了3个按钮，第一个是“添加”按钮，因为在修改学生信息时会把它上面的文字改为“修改”，所以“清空”按钮也要在执行reset()之后把它的文字改为“添加”才行。

12.8　删除指定的学生信息

“删除”按钮是表单上的第三个按钮，它的作用是删除当前显示的学生信息。既然是从数据库中删除已有的学生信息，那它就只能在“修改”信息的状态下起作用。因为在没有获得学生信息的id之前，无法执行删除操作。

“删除”按钮对应的handler处理函数如代码清单12-15所示。

代码清单12-15　删除学生信息的处理函数

```
{
text: '删除',
    handler: function() {
        var id = form.getForm().findField('id').getValue();
        if (id == '') {
            Ext.Msg.alert('提示', '请选择需要删除的信息。');
        } else {
            Ext.Ajax.request({
                url: './jsp/remove.jsp',
                success: function(response) {
                    var json = Ext.decode(response.responseText);
                    if (json.success) {
                        Ext.Msg.alert('消息', json.msg, function() {
                            grid.getStore().reload();
                            form.getForm().reset();
                            form.buttons[0].setText('添加');
```

```
                        });
                    }
                },
                failure: function() {
                    Ext.Msg.alert('错误', "删除失败");
                },
                params: "id=" + id
            });
        }
    }
}
```

处理函数首先要判断form.getForm().findField('id').getValue()的值是否为空，如果id为空，表示还没有选择需要删除的信息，这时会弹出提示信息，同时中止删除操作。

如果id不为空，表示已经选择了需要删除的信息，这样可以把对应信息的id发送到后台执行删除操作。与"添加"按钮不同的是，向后台发送要删除的信息的id时要通过Ext.Ajax来实现，因为删除操作并不需要将表单中所有输入组件的数据都发送到后台。

使用Ext.Ajax.request()函数发送信息时，url参数表示发送给后台的jsp/remove.jsp脚本进行处理，params表示将学生信息的id作为参数传递给后台。success对应的处理方法会在响应成功时执行，与表单不同的是，这里不再需要在响应的JSON中设置{success:true}，但我们需要使用Ext.decode()函数将响应返回的response.responseText字符串手工转换为JSON对象，然后用Ext.Msg.alert()显示响应中的提示信息。响应成功后，刷新表格数据，清空表单内容的操作与之前的添加和修改操作相同。

因为删除操作只向后台的remove.jsp发送需要删除的信息的id，所以remove.jsp中的处理代码也比较简单，如代码清单12-16所示。

代码清单12-16　remove.jsp

```
<%@ page contentType="application/json;charset=utf-8" import="com.family168.student.
*" %><%
    request.setCharacterEncoding("utf-8");
    response.setCharacterEncoding("utf-8");

    String id = request.getParameter("id");
    StudentDao dao = StudentDao.getInstance();
    dao.remove(Long.parseLong(id));

    out.print("{success:true,msg:'删除成功'}");
%>
```

这里我们还是先设置请求和响应的编码，然后从请求中获得参数id的值，转换成长整型（long），调用StudentDao的remove()方法执行删除操作。如果处理的过程没有出现异常，就说明删除操作成功，最后向响应中写入删除成功的提示信息。

这样我们就完成了删除指定学生信息的操作。

12.9　在表格和表单之间进行数据交互

我们在上面已经实现了在表格中显示学生信息列表，也实现了使用表单对学生信息执行添

加、修改和删除等操作。但是，表格和表单之间的数据还无法交互使用。

一个尚未解决的问题是，如何在表单中显示我们需要修改的学生信息？这个问题也适用于删除操作，如果我们不提供选择某一条学生记录的方法，那么修改和删除操作都无法进行。

在这个示例中，我们希望在单击左侧的表格时同步更新右边表单中的数据。如果用户单击表格中的某一行，就会把这行对应的学生信息放到表单中显示，于是我们就能对这条信息进行修改和删除操作了。为此，我们要给表格添加一个事件监听函数，专门处理鼠标点击事件，如下面的代码所示：

```
// 单击修改信息开始
grid.on('rowclick', function(grid, rowIndex, event) {
    var record = grid.getStore().getAt(rowIndex);
    form.getForm().loadRecord(record);
    form.buttons[0].setText('修改');
});
// 单击修改信息结束
```

这里监听的事件名为rowclick，它对应Ext.grid.RowSelectionModel的监听事件，每当用户选中表格中的一行时，就会触发该事件。事件被触发的同时还会执行我们设置的监听函数。

监听函数预设了3个参数：第一个参数grid表示哪个表格被点击了；第二个参数rowIndex表示选中了哪一行；event是EXT内部通用的事件对象，我们在这里没有用到。

在单击事件被触发，执行对应的监听函数时，首先通过grid.getStore().getAt(rowIndex);获得被选中的这一行对应的record。这个record是保存在store中的数据，表格上没有显示出来的id也包含在其中。对应的所有学生信息都可以从这个record中获得，但并不需要从record中把学生信息逐一取出来，然后再逐一放到表单中。表单提供的loadRecord()函数可以一次性将record中的数据赋予表单中的输入组件，只要保证输入组件的name或hiddenName与record中定义的属性一致即可。

在使用loadRecord()将表格中选择的数据复制到表单中以后，我们再调用form.buttons[0].setText('修改');将表单中的第一个按钮的文字设置为“修改”。这样用户就知道现在提交表单执行的是对某一条学生信息进行修改的操作。如果要继续添加新的学生信息，可以单击“清空”按钮，它会将刚刚从表格中复制的信息都清除掉，包括id隐藏域中的数据，还会把第一个按钮上的“修改”设置为“添加”。再次输入数据并单击“提交”按钮，这时执行的就是“添加”操作了。

到此为止，我们用前面学过的知识实现了一个完整的学生信息管理系统。其中涉及BorderLayout的布局应用、表格的分页显示和数据排序、表单的提交和清空、利用Ajax与后台进行数据交互、通过事件监听实现表格与表单之间的数据交互等知识。

这个示例虽然简单，但基本上包含了所有的常见操作，可以供大家练习时参考。

12.10 提升加载速度

在本节中，我们将基于前面介绍的学生信息管理系统，讨论一下如何提升EXT系统的加载速度。

在人们讨论EXT的性能问题时，首先担心的就是ext-all.js这个脚本文件的体积，EXT已经对

所有源代码进行了压缩和混淆，它只是压缩混淆之前的版本ext-all-debug.js大小的1/4，ext-all.js的大小是610 KB，ext-all-debug.js的大小是2341 KB。我们从这一点就可以看到EXT已经在应用性能的提升方面做出的努力了。

可是很多人对这3/4的体积减小并不满意，他们还期望继续减小网络传输的数据量，以实现更短的传输时间，让应用能够更快地呈现在用户面前。下面我们来介绍几种简单易用的调优方法，借助这些方法可以实现在不影响整体功能的基础上提升应用的加载速度。

12.10.1 对 JavaScript 文件进行压缩混淆

我们建议在项目开发完成之后，系统上线之前，使用工具将项目中的JavaScript文件整合为一个，并进行压缩，去除JavaScript文件中的注释、多余换行和空格符。再对JavaScript文件进行语法混淆，进一步减小文件的大小。

这里选用yuicompressor对JavaScript进行压缩和混淆。yuicompressor是YUI官方提供的JavaScript压缩工具，EXT也使用它对源码进行压缩与混淆的操作。

我们通过命令行执行yuicompressor，执行命令如下：

```
java -jar yuicompressor-2.4.2.jar student.js -o student-min.js --charset UTF-8
```

这样就会将原来项目中的student.js文件进行压缩混淆，生成student-min.js文件，因为student.js中包含中文，所以还要使用--charset指定源文件的编码为UTF-8，否则中文在压缩后会变成乱码。比较student-min.js和student.js可以看到，压缩前student.js的大小为8 KB，压缩后student-min.js的大小为4 KB，减小了50%的体积。随着项目中JavaScript文件总数的增加，这个压缩比率还会继续上升，可以说对项目中的JavaScript文件进行压缩混淆是最行之有效的一种提升性能的方法。

注意在进行JavaScript文件的压缩和混淆时要保留源文件，执行过压缩混淆之后的JavaScript文件内容随着本身体积的减小，也会变得不适合后期的开发维护，要避免压缩混淆工具毁坏源码的情况。

下面是压缩后student-min.js中内容的一部分，从中可以看到代码已经变得难以阅读了，代码如下所示：

```
Ext.onReady(function(){Ext.QuickTips.init();var g=function(h){if(h==1){
```

12.10.2 使用客户端缓存

客户端缓存是另一种常用的用来提升性能的手段，设置正确的话，可以实现所有静态资源只在用户第一次访问应用时进行加载，之后的请求都会利用本地缓存，不需重复下载相同的静态资源。

客户端缓存主要是利用了HTTP协议中的304 Not Modified这个响应状态，当客户端请求的资源在服务器端未进行修改时，服务器就会返回304 Not Modified响应状态，这时客户端就会直接利用本地缓存。

我们可以使用Firebug查看访问应用时客户端请求资源以及服务器响应的状态。

当客户端第一次访问应用时，请求资源如图12-6所示。

从图12-6中可以看到，第一次访问应用时的响应状态都为200 OK，从请求到完成加载ext-all.js总共用了905 ms。

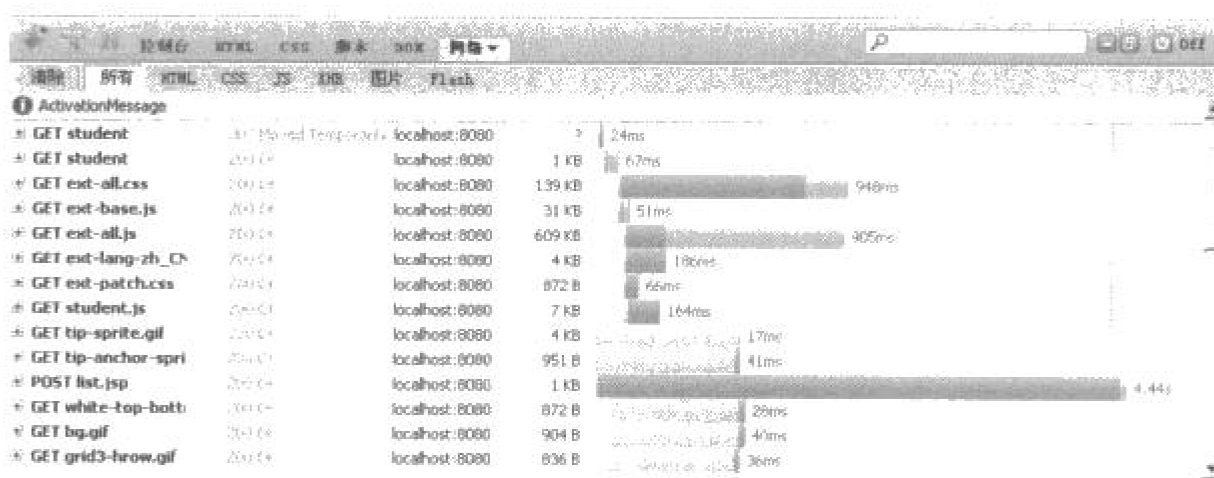


图12-6　第一次访问应用时的请求状态

如果现在刷新一下页面，再次访问应用，请求和响应就会变成如图12-7所示的状态。

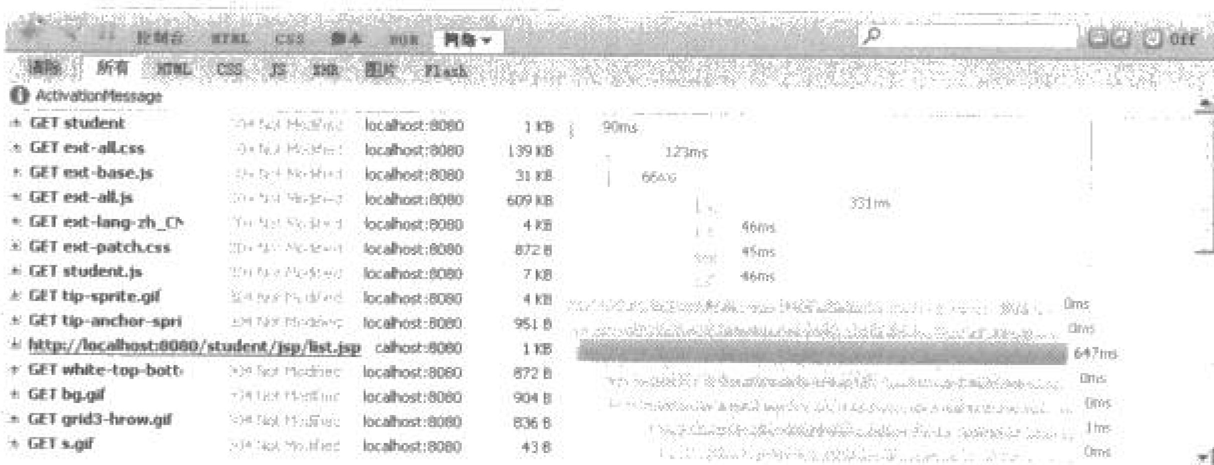


图12-7　再次访问应用时的请求状态

从图12-7中可以发现，再次访问应用时的响应状态为304 Not Modified，因为利用了本地缓存，ext-all.js从请求到完成加载总共用了331 ms。

上面的比较只是将使用客户端缓存前后的情况进行了简单的比较，实际的测试数据会因为网络情况的不同而出现改变，但可以肯定的是利用了客户端缓存之后，浏览器便不再需要在每次请求时都去服务器上下载ext-all.js和其他已经获得的资源了，因此也不会出现因为ext-all.js体积过大拖慢网络传输速度，从而影响系统响应效率的问题。

默认情况下，浏览器和服务器都会开启对本地缓存功能的支持，所以一般不需要再去进行额外的配置，在需要精确控制缓存周期时可以参考实际使用服务器提供的文档进行相应的配置。

12.10.3　使用 GZIP 压缩

我们上面提到了使用工具对JavaScript文件进行压缩混淆，进而实现在系统加载时传输较少的数据量。只是即便EXT对所有代码进行了压缩混淆，最终的ext-all.js体积依然有610 KB，如何才能进一步减小传输的数据量呢？这时我们可以选择使用GZIP对JavaScript文件进行进一步的压缩。

目前市场上的主流浏览器都支持对GZIP格式的文件进行处理。服务器将GZIP压缩过的文件传输给浏览器，浏览器在接收到数据之后，首先进行解压缩将文件还原为压缩前的形式，然后再

进行加载。这种方式可以减小网络传输数据量，对浏览器端造成的性能影响也能保证在可接受的范围之内。

首先使用工具将ext-all.js压缩为GZIP格式，压缩后获得文件名为ext-all.gz.js，现在可以比较一下两者体积的大小。未使用GIP压缩的ext-all.js文件大小为610 KB，使用了GZIP压缩的ext-all-gz.js大小为167KB，又减小了3/4。

下面将index.jsp中原来引用的ext-all.js部分代码替换为ext-all.gz.js，代码如下所示：

```
<script type="text/javascript" src="ext-3.0.0/ext-all.gz.js"></script>
```

接下来还需要告诉浏览器，ext-all.gz.js使用了GZIP格式，这样浏览器才会在接收到数据之后对ext-all.gz.js进行解压。为此我们新建了一个GzipFilter.java，它会在用户请求ext-all.gz.js时，向响应中添加一个头信息，告知浏览器需要使用GZIP对ext-all.gz.js进行处理。

GzipFilter.java代码内容如下所示：

```
package com.family168.student;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class GzipFilter implements Filter {
    public void init(FilterConfig filterConfig)
          throws ServletException {
    }

    public void destroy() {
    }

    public void doFilter(ServletRequest request,
                    ServletResponse response,
                    FilterChain chain)
              throws IOException,
                    ServletException {

        String url = ((HttpServletRequest) request).getServletPath();
        if (url.endsWith(".gz.js")) {
            ((HttpServletResponse)response).setHeader("Content-Encoding", "gzip");
        }

        chain.doFilter(request, response);
    }
}
```

这样，GzipFilter会在请求URL以.gz.js结尾时，自动向请求中添加内容为"Content-Encoding"，"gzip"的头信息，这样浏览器就会知道如何处理对应的信息内容了。

为了使GzipFilter生效，我们还需要修改web.xml，在其中添加过滤器映射，代码内容如下：

```
<filter>
    <filter-name>gzip-filter</filter-name>
    <filter-class>com.family168.student.GzipFilter</filter-class>
</filter>

<filter-mapping>
```

12

```
        <filter-name>gzip-filter</filter-name>
        <url-pattern>*.js</url-pattern>
    </filter-mapping>
```

这里我们将GzipFilter配置为监听所有以.js结尾的请求，但是实际上它只会处理以.gz.js结尾的请求。

经过上面的配置之后，我们就可以访问应用，查看使用GZIP压缩JavaScript的效果了，从Firebug中可以看到图12-8中的信息。



图12-8　使用GZIP压缩JavaScript后的响应效果

从图12-8中可以看到，我们请求的ext-all.gz.js只有167 KB，对应的响应头信息中包含了Content-Encoding gzip的内容，这样浏览器就知道应该使用GZIP格式处理获得的内容了。

在使用GZIP压缩ext-all.js之后，在FireFox 3.5与IE 7下均测试通过，应用运行正常。

除了以上几种提升加载效率的方式，很多人还希望对EXT进行裁剪，认为将不需要的功能删除掉就可以减小网络请求的时耗。但是我们并不推荐使用这种方式，因为EXT中的继承模型比较复杂，即使删除一些组件也不能明显减小ext-all.js的大小，而且容易在系统后期进行功能扩充或系统升级时造成一些无可预见的问题，使用上面介绍的GZIP压缩方法已经可以将ext-all.js压缩到167KB，基本可以满足大部分系统调优的情况。

不过，如果还是有读者对裁剪ext-all.js有兴趣的话，可以参考EXT官方提供的对应工具，它可以根据自己所需的组件生成自定义的JavaScript文件。不过目前免费版的工具只支持2.3.0和1.1.1两个版本的EXT，网址为http://www.extjs.com/products/extjs/build/。

12.11　小结

本章详细演示了如何实现一个学生信息管理系统，其中简要介绍了如何使用Java访问数据库，以及一些SQL语句的使用方法。重点介绍了如何使EXT与后台进行交互，实现对数据库信息进行分页显示和后台排序。此外，还讲解了如何在EXT中对数据库信息执行添加、删除、修改和查找等基本操作，并提出了一种表格与表单之间进行数据交互的方式。

最后基于完成的学生信息管理系统介绍了提升系统加载效率的一些常用方法，包括使用工具将JavaScript文件进行压缩混淆、利用客户端缓存和使用GZIP压缩。

第13章 复杂实例

本章内容
- VIP客户统计系统
- Tracker任务跟踪系统

本章将演示两个稍显复杂的例子：VIP客户统计系统和Tracker任务跟踪系统。这两个系统虽然规模不大，但是包含了数据分类查找、统计报表等功能。功能中还使用到了一些扩展件，希望可以起到抛砖引玉的作用，让大家在前面的基础上对Ext JS有更进一步的了解。

13.1 VIP客户统计系统

这是一个简易的信息统计查询工具，它甚至没有服务器端的代码，完全依靠JavaScript提供各种数据。在这个小系统中我们可以分类查看不同客户的信息，以及由这些信息汇总的图形报表。

用户单击index.html就可以运行VIP客户统计系统，首页效果如图13-1所示。主页面分为左右两个部分，左侧是操作选项，可以根据客户的类型或者是否过期，分类显示对应的客户信息以及统计报表。右侧显示客户的详细信息，包括称呼、开始时间、结束时间、时间线、类型、邮箱和QQ号码。

该系统中最主要的功能就是如何根据用户类型，在页面右侧的表格中显示对应的用户信息。由于系统没有后台支持，所有的数据都保存在内存中，所以只能设法过滤内存中已有的数据，达到分类显示的效果。

我们将整个项目按功能划分为如下几个部分。
- fn.js，包含可复用的功能函数。
- pay.filter.js，扩展MemoryProxy，使其支持分页和数据过滤功能。
- pay.grid.js，实现页面右侧的表格。
- pay.tree.js，实现页面左侧的树形。
- pay.js，负责整合其他功能，定义程序入口。
- test.pay.data.js，保存测试数据。

图13-1 VIP客户统计系统（用户列表）

项目的入口点位于pay.js。页面加载完毕后，会调用它内部的`Ext.onReady(function(){});`函数。这个函数会创建左侧的树形和右侧的表格，再将两者拼合在统一的Viewport布局中，最后渲染到页面上，形成大家看到的效果。

对应的功能代码如代码清单13-1所示。

代码清单13-1 初始化树形和表格，设置页面布局

```
// 表格
var grid = Pay.createGrid();
// 树形
var tree = Pay.createTree();

// 布局开始
var tabPanel = new Ext.TabPanel({
    region: 'center',
    activeTab: 0,
    items: [grid]
});

var viewport = new Ext.Viewport({
    layout:'border',
    items:[tabPanel, tree, {
        region: 'north',
        contentEl: 'north'
    },{
        region: 'south',
        contentEl: 'south'
    }]
});
// 布局结束
```

Pay.createGrid()和Pay.createTree()分别定义在pay.grid.js和pay.tree.js中，分别返回系统的左侧树形菜单和右侧表格。使用这种方式，我们可以把整个系统切分成不同的功能模块，分散到多个JavaScript进行开发。

左侧树形采用的加载方式是给TreePanel设置TreeLoader为空的参数，让它去加载root中的children属性。这样可以直接在根节点中使用JSON格式的数据定义树形节点，就不必使用烦琐的new Ext.tree.TreeNode创建各个节点的实例了，如代码清单13-2所示。

代码清单13-2 左侧树形菜单

```
Ext.namespace("Pay");

Pay.createTree = function() {

    var tree = new Ext.tree.TreePanel({
        region: 'west',
        title: '搜索类型',
        width: 150,
        slide: true,
        loadMask: true,
        loader: new Ext.tree.TreeLoader()
    });
    var root = new Ext.tree.AsyncTreeNode({
        id: 'config',
        text: '选项',
        children: [{
            id: 'level',
            text: '用户类型',
            children: [{
                id: 'allLevel',
                text: '全部',
                leaf: true
            },{
                id: 'noSupport',
                text:'无支持',
                leaf: true
            },{
                id: 'month',
                text: '包月',
                leaf: true
            },{
                id: 'year',
                text: '包年',
                leaf: true
            },{
                id: 'always',
                text:'终身',
                leaf: true
            }]
        },{
            id: 'outOfDate',
            text: '是否过期',
```

13

```
                children: [{
                    id: 'allOutOfDate',
                    text: '全部',
                    leaf: true
                },{
                    id: 'notOutOfDate',
                    text: '未过期',
                    leaf: true
                },{
                    id: 'alreadyOutOfDate',
                    text: '已过期',
                    leaf: true
                }]
            },{
                id: 'report',
                text: '统计图表',
                children: [{
                    id: 'levelReport',
                    text: '按用户类型',
                    leaf: true
                },{
                    id: 'outOfDateReport',
                    text: '按是否过期',
                    leaf: true
                }]
            }]
        });

        tree.setRootNode(root);

        return tree;
    };
```

pay.js中为左侧树形添加了事件，这样在单击左侧树形菜单时，右侧表格就会进行联动修改，如果点击了用户类型和用户是否过期，就会刷新右侧表格中的数据，显示选中类型的用户信息。如果点击报表，就会在右侧TabPanel中添加新的报表页面。

对应代码如代码清单13-3所示。

代码清单13-3 为左侧树形菜单设置监听事件

```
// 树形点击事件
tree.on('click', function(node) {
    var id = node.id;
    if (id == 'allLevel' || id == 'allOutOfDate') {
        grid.getStore().baseParams.filter = undefined;
    } else if (id == 'noSupport') {
        grid.getStore().baseParams.filter = {id: 'level', value: 0};
    } else if (id == 'month') {
        grid.getStore().baseParams.filter = {id: 'level', value: 1};
    } else if (id == 'year') {
        grid.getStore().baseParams.filter = {id: 'level', value: 2};
    } else if (id == 'always') {
        grid.getStore().baseParams.filter = {id: 'level', value: 3};
```

```
    } else if (id == 'notOutOfDate') {
        grid.getStore().baseParams.filter = {id: 'notOutOfDate', value: notOutOfDate};
    } else if (id == 'alreadyOutOfDate') {
        grid.getStore().baseParams.filter = {id: 'alreadyOutOfDate', value: isOutOfDate};
    }
    if (id == 'levelReport') {
        var tabItem = tabPanel.getItem('chart-level');
        if (tabItem == null) {
            tabItem = tabPanel.add(new Ext.ux.IFrameComponent({
                id: 'chart-level',
                title: '用户等级统计图表',
                closable: true,
                url: 'chart-level.html'
            }));
        }
        tabPanel.activate(tabItem);
    } else if (id == 'outOfDateReport') {
        var tabItem = tabPanel.getItem('chart-outofdate');
        if (tabItem == null) {
            tabItem = tabPanel.add(new Ext.ux.IFrameComponent({
                id: 'chart-outofdate',
                title: '是否过期统计图表',
                closable: true,
                url: 'chart-outofdate.html'
            }));
        }
        tabPanel.activate(tabItem);
    } else {
        tabPanel.activate(0);
        grid.getStore().reload();
    }
});
tree.getRootNode().expand(true);
```

所有VIP用户信息都保存在test.pay.data.js中，用户信息的数据格式如代码清单13-4所示。

代码清单13-4 VIP客户数据

```
Ext.namespace("Pay");

Pay.data = {
    totalProperty: 21,
    root: [{
        "id":1,
        "name":"尖叫的土豆",
        "startDate":null,
        "endDate":null,
        "level":3,
        "email":"forgetdavi@gmail.com",
        "qq":"1184282",
        "descn":"<hr>感谢大家支持我们的《深入浅出Ext JS》<br>希望大家多给我们提出宝贵意见<br>谢谢。"
    },{
        "id":2,
        "name":"临远",
```

13

```
            "startDate":null,
            "endDate":null,
            "level":3,
            "email":"xyz20003@gmail.com",
            "qq":"416529445",
            "descn":""
        },{
            "id":3,
            "name":"WT WT W!",
            "startDate":null,
            "endDate":null,
            "level":3,
            "email":"kaiyuanlthh@gmail.com",
            "qq":"289298692",
            "descn":""
        }
    ]};
```

首先使用Ext.namespace()为项目定义一个自己的命名空间，之后就可以在Pay这个命名空间下进行开发，这样不容易与其他模块发生冲突。Ext.namespace()不会对已经存在的命名空间造成破坏，可以放心使用。

每个客户都拥有相同的设置，包括id、name、startDate、endDate、level、email、qq和descn(备注)。其中level可能为:0（无在线支持）、1（包月）、2（包年）和3（终身）4种类型，它们会被渲染为不同样式的文字，最终显示在表格中。

图13-2只显示包月类型的用户信息。

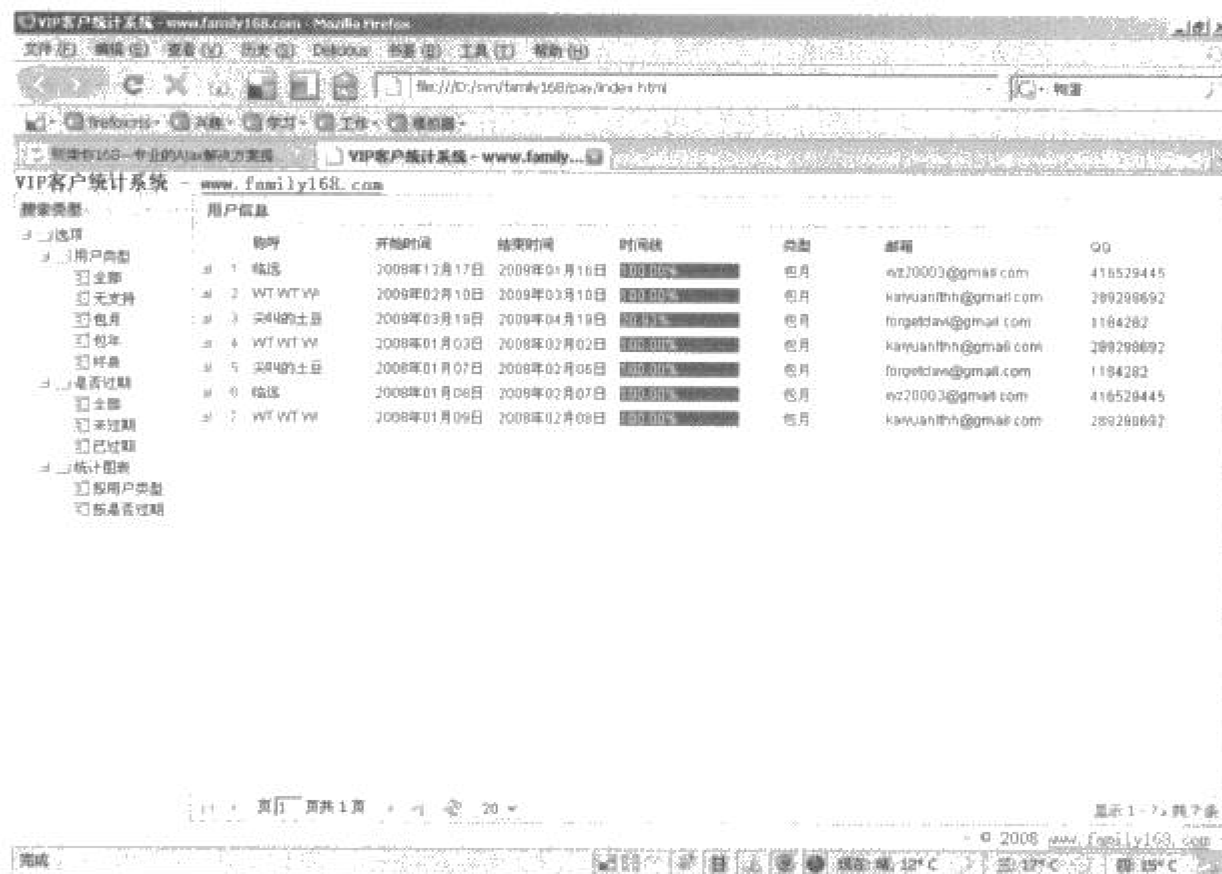


图13-2　VIP客户统计系统（包月用户列表）

大家可能已经注意到表格中时间线那一列了，这一列使用不同颜色来提示用户的VIP期限已经过去了多长时间。它与“类型”列一样，都是在ColumnModel中设置对应的renderer渲染函数来实现的。我们根据当前时间和用户开始时间与结束时间进行比较，使用DIV标签配合CSS样式模拟了一个简易的进度条。

时间线使用的renderer函数在pay.grid.js中，如代码清单13-5所示。

代码清单13-5 时间线使用的renderer函数

```
{
    id: 'timeline',
    header: '时间线',
    renderer: function(value, cellmeta, record) {
        var level = record.get('level');
        // 无支持
        if (level == 0) {
            return "<div style='background-color:#CCCCCC;width:100px;'> </div>";
        } else if (level == 3) {
            return "<div style='background-color:lightgreen;width:100px;'> </div>";
        } else {
            var startDate = record.get('startDate');
            var endDate = record.get('endDate');
            var total = endDate.getTime() - startDate.getTime();
            var current = new Date().getTime() - startDate.getTime();
            var percent = current / total * 100;
            if (percent > 100) {
                percent = 100;
            }

            var p = Ext.util.Format.usMoney(percent).substring(1);
            return "<div style='background-color:green;color:white;width:100px;'>" +
                   "<div style='background-color:red;width:" + percent + "px;'>" + p +
                   "%</div>" +
                  "</div>";
        }
    },
    width: 80
}
```

我们使用了renderer函数的3个参数。

- 第一步，从record中取得level用户类型。当用户类型为无支持时，直接显示为宽度100的红色进度条，表示已经过期。
- 第二步，如果用户类型为终身，显示宽度100的浅绿色进度条，表示从不过期。
- 第三步，如果用户类型为其他，即包月或包年时，获得record中的startDate和endDate，再根据当前时间计算百分比，使用Ext.util.Format.usMoney()函数对获得的百分比进行格式化，最终显示一条绿色背景、红色前景的进度条。

从代码清单13-5中可以看到，我们只使用了renderer中最基本的操作，通过record参数获得该行数据，并进行操作输出对应的HTML信息，大家在此基础上也可以考虑使用图片等资源绘制出更加漂亮的效果。

从pay.grid.js中可以看到，为了实现行列显示、渲染功能，我们使用了大量的代码，而实际上实现系统关键业务的代码只有一行：`var proxy = new Ext.ux.FilterPagingMemoryProxy(Pay.data);`。创建的`Ext.ux.FilterPagingMemoryProxy`实例将从`Pay.data`这个变量中提取数据，并将封装好的数据提供给表格使用。`Ext.ux`是EXT为用户扩展（User Extend）专门预留的命名空间，我们不需要使用`Ext.namespace()`就可以直接使用。`Ext.ux.FilterPagingMemoryProxy`的代码可以在pay.filter.js中找到，代码中最主要的部分是对原始数据进行过滤，即根据给定的条件对数据进行过滤，只保留符合条件的数据，如代码清单13-6所示。

代码清单13-6　FilterPagingMemoryProxy中的数据过滤部分

```
if (params.filter !== undefined) {
    var filter = params.filter;
    var id = filter.id;
    var value = filter.value;
    if (id == 'level') {
        result.records = result.records.filter(function(record){
            return String(record.data[id]).match(value) ? true : false;
        });
    } else if (id == 'notOutOfDate') {
        result.records = result.records.filter(function(record){
            return notOutOfDate(record);
        });
    } else if (id == 'alreadyOutOfDate') {
        result.records = result.records.filter(function(record){
            return isOutOfDate(record);
        });
    }
    result.totalRecords = result.records.length;
}
```

`FilterPagingMemoryProxy`会判断`params`中是否存在`filter`参数，如果存在就要对数据进行过滤。如果`id`为`level`，则需要根据用户类型进行数据过滤，这时调用`records`的`filter()`函数和`String`的`match()`函数判断哪些数据符合过滤条件。如果`id`为`notOutOfDate`或`alreadyOutOfDate`，则需要根据用户是否过期进行数据过滤，这时调用fn.js中提供的`notOutOfDate()`和`isOutOfDate()`两个函数判断用户是否已经过期，以此来决定显示哪些数据。

这样我们只需要设置`grid.getStore().baseParams.filter = {id: 'level', value: 0};`再调用`grid.getStore().reload();`函数，就可以让表格根据我们设置的条件进行过滤了。

除了`FilterPagingMemoryProxy`这个自定义的扩展件之外，我们还在表格中使用了两个官方提供扩展件：`Ext.ux.PageSizePlugin.js`和`RowExpander.js`。

`Ext.ux.PagingSizePlugin`用于修改表格中每页显示的最大信息数。使用的时候可以直接把它添加到`bbar`中，如代码清单13-7所示。

代码清单13-7　为表格添加PageSizePlugin插件

```
bbar: new Ext.PagingToolbar({
    pageSize: pageSize,
    store: store,
```

```
        displayInfo: true,
        plugins:[new Ext.ux.PageSizePlugin()]
    })
```

RowExpander会在表格每行前面添加一个小图标，点击这个图标可以将这一行展开，展开的内容会显示在这一行的下方，我们可以使用Template自由设置展开的内容，如代码清单13-8所示。

代码清单13-8 为表格添加RowExpander插件

```
var expander = new Ext.grid.RowExpander({
    tpl : new Ext.Template(
        '<p>{descn}</p><br /><br />'
    )
});
var cm = new Ext.grid.ColumnModel([
    expander
]);
var grid = new Ext.grid.GridPanel({
    sm: new Ext.grid.RowSelectionModel({
        selectRow: function(index, keepExisting, preventViewNotify) {
            expander.toggleRow(index);
        }
    }),
    plugins: expander
});
```

PageSizePlugin和RowExpander的显示效果如图13-3所示。

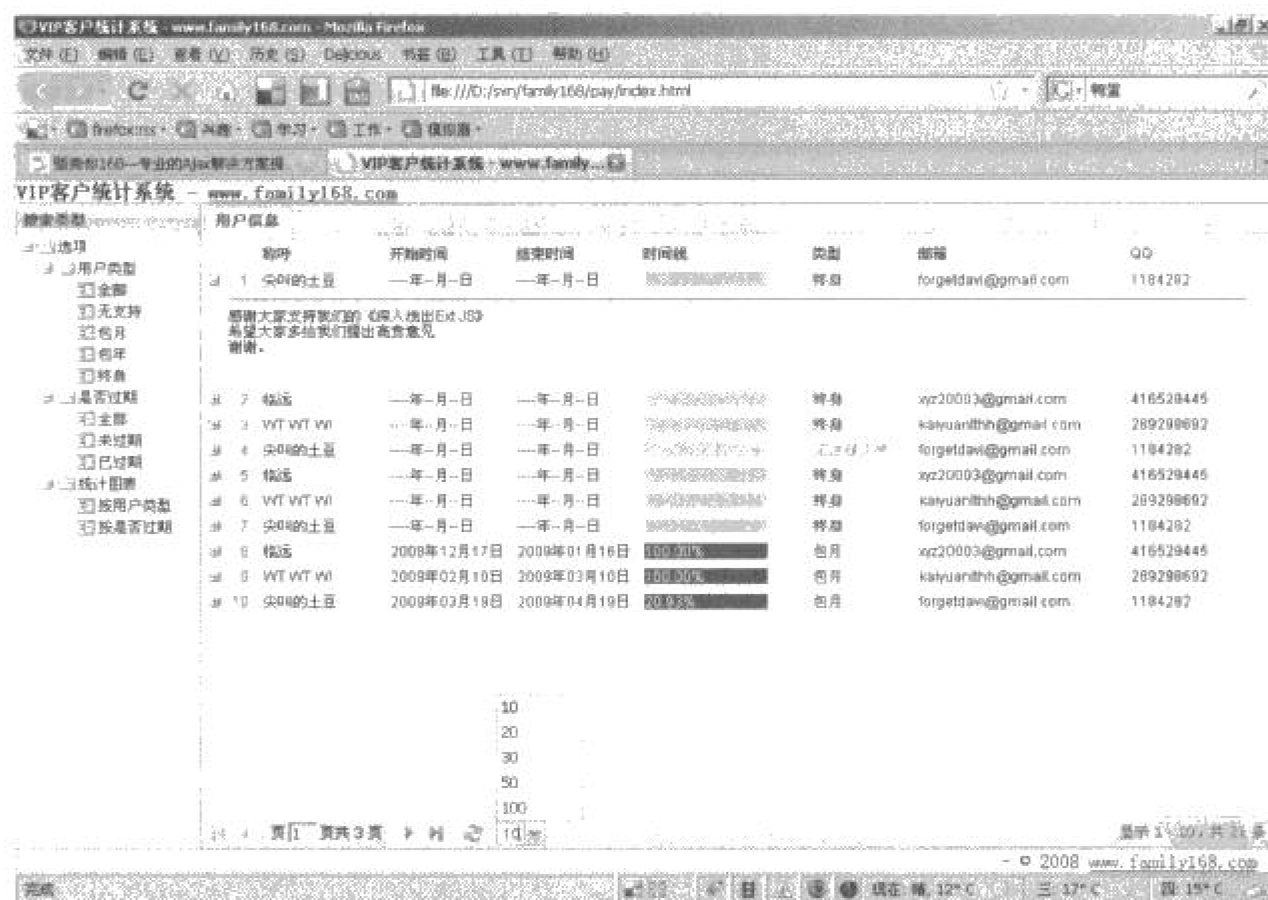


图13-3 使用PageSizePlugin和RowExpander的显示效果

系统的最后一项功能是统计报表。我们可以按照用户类型和是否过期生成两种统计报表，报表图形并不是使用EXT实现的，不过在显示报表页面的时候使用了iframe。这样做的好处是不用将所有代码都加载到首页中，虽然RIA宣扬one page one application，但是使用iframe可以在一定程度上避免一次加载过多的资源文件，在实际中依然拥有适用的场景。

为了方便起见，我们使用了EXT官方提供的Ext.ux.IFrameComponent.js来实现在页面中插入iframe的功能，对应代码如代码清单13-9所示。

代码清单13-9　使用IFrameComponent向页面中插入iframe

```
tabItem = tabPanel.add(new Ext.ux.IFrameComponent({
    id: 'chart-outofdate',
    title: '是否过期统计图表',
    closable: true,
    url: 'chart-outofdate.html'
}));
```

使用iframe的统计报表，显示效果如图13-4所示。

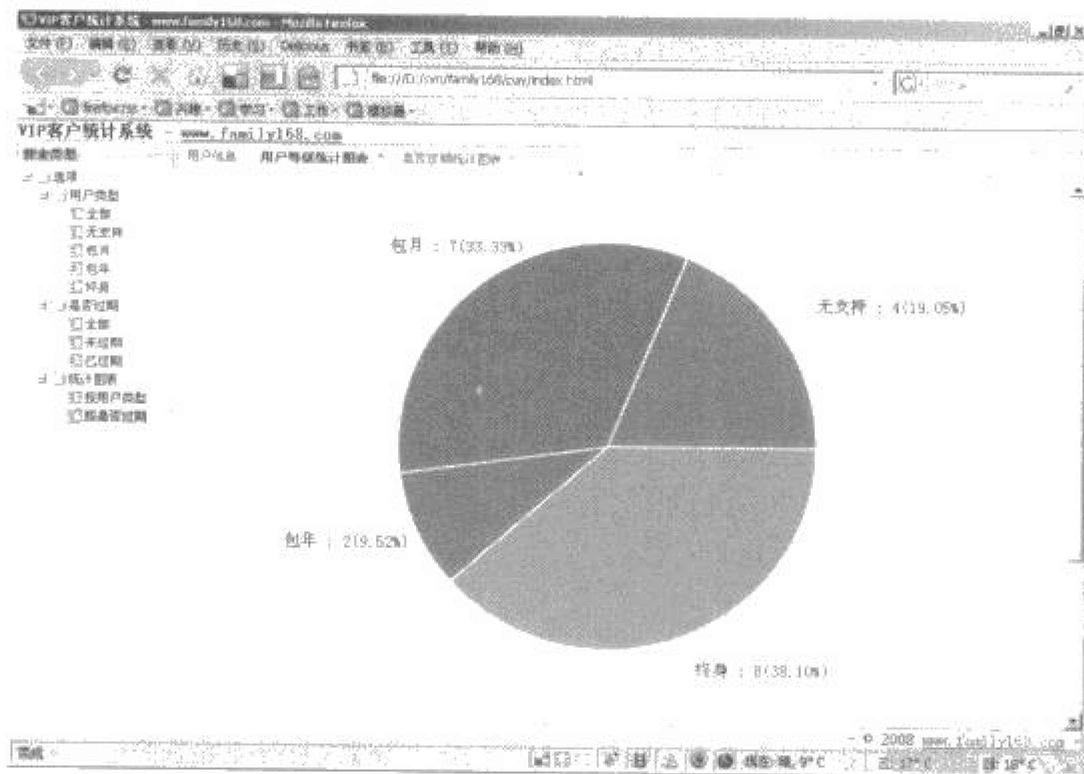


图13-4　VIP客户统计系统（用户等级统计图表）

13.2　Tracker 任务跟踪系统

Tracker是一个简易的任务跟踪系统，它使用了最基本的SSH（Struts.Spring.Hibernate）开发框架，通过嵌入式数据库Hsqldb保存数据，依靠Maven 2管理项目流程。这俨然已经是一个小而全的企业系统了。

EXT在系统中负责前台展示的部分，后台通过Struts 2结合json-lib与前台的EXT进行交互。在开发过程中，我们封装了JsonGrid和JsonTree这些基本组件，很大程度上减少了编码的数量，

提高的开发效率。

系统界面如图13-5所示。

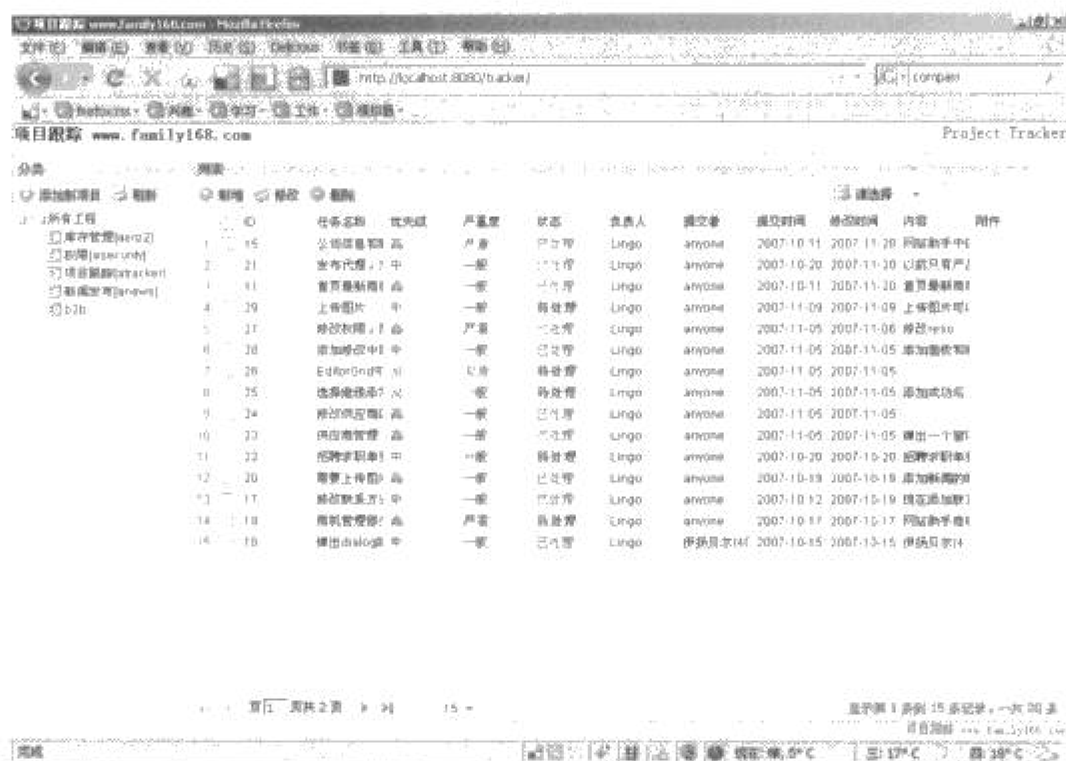


图13-5 Tracker任务跟踪系统

系统左侧是以JsonTree为基础生成的树形菜单，显示了所有工程的信息，我们可以直接在左侧面板部分进行添加、修改、删除等操作。

进行详细配置后，右键功能菜单效果如图13-6所示。

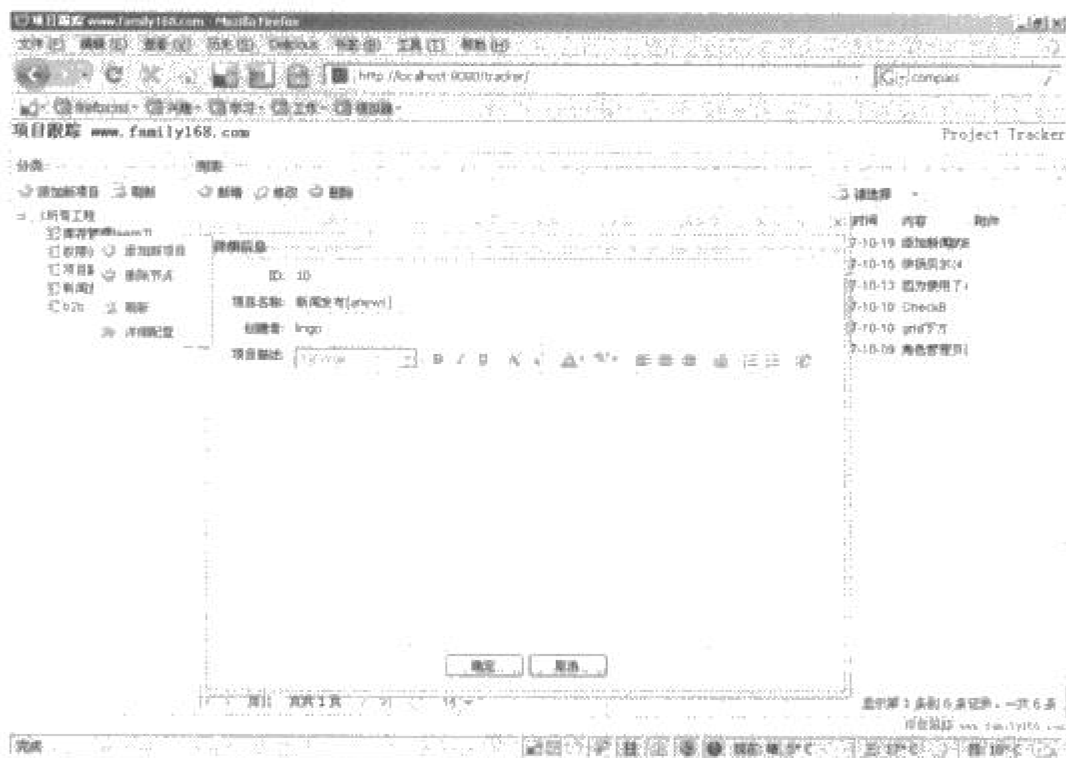


图13-6 维护左侧树形菜单

创建左侧树形菜单的代码如代码清单13-10所示。

代码清单13-10 创建左侧树形菜单

```
var metaData = [
    {name: 'id',      fieldLabel: "ID",      allowBlank: true},
    {name: 'name',    fieldLabel: "项目名称", allowBlank: false},
    {name: 'founder', fieldLabel: "创建者",   allowBlank: false},
    {name: 'summary', fieldLabel: "项目描述", allowBlank: true, xtype: 'htmleditor'}
];
Tracker.tree = new TrackerTree({
    formConfig: metaData,
    rootName: '所有工程',
    dlgWidth: 450,
    dlgHeight: 250,
    urlGetAllTree: "./trackerProject!getAllTree.do",
    urlInsertTree: "./trackerProject!insertTree.do",
    urlRemoveTree: "./trackerProject!removeTree.do",
    urlSortTree: "./trackerProject!sortTree.do",
    urlLoadData: "./trackerProject!loadData.do",
    urlUpdateTree: "./trackerProject!updateTree.do",
    region: 'west',
    title: '分类',
    width: 169,
    minSize: 169,
    split: true
});
```

TrackerTree直接继承自JsonTree，它在继承了JsonTree所有功能的基础上，对内部功能进行了一些自定义修改。我们来看一下它所使用的初始化参数，以便对它有更进一步的了解。

- formConfig，新建或更新时生成表单窗口的配置，可以使用xtype指定编辑器的类型。
- rootName，树形的根节点。
- dlgWidth和dlgHeigth用来设置生成表单窗口的宽和高。
- urlGetAllTree，从后台获取树形数据的地址。
- urlInsertTree，添加节点时请求的后台地址。
- urlRemoveTree，删除节点时请求的后台地址。
- urlSortTree，对节点进行排序时请求的后台地址。

这样我们不需要再另外写其他代码，只使用new关键字创建了一个TrackerTree的实例，就直接拥有了CRUD（增删改查）这些基本操作。如果项目中有多个操作相似的树形需要创建，那么这种继承超类的方式无疑是一种省时省力的好方法。

如果需要修改JsonTree中的功能，可以像TrackerTree一样，使用Ext.extend()继承JsonTree生成一个子类，在子类中进行扩展或修改超类中的功能。如代码清单13-11所示。

代码清单13-11 使用Ext.extend()继承JsonTree生成子类

```
TrackerTree = Ext.extend(Ext.lingo.JsonTree, {

    // 生成工具条
```

```
    buildToolbar: function() {
        var toolbar = [{
            text     : '添加新项目',
            iconCls : 'add',
            tooltip : '添加选中节点的下级分类',
            handler : this.createNewProject.createDelegate(this)
        }, {
            text     : '刷新',
            iconCls : 'refresh',
            tooltip : '刷新所有节点',
            handler : this.refresh.createDelegate(this)
        }];
        return toolbar;
    },

    // 生成右键菜单
    buildContextMenu : function() {
        this.on('contextmenu', this.prepareContext);
        this.contextMenu = new Ext.menu.Menu({
            id     : 'copyCtx',
            items : [{
                id       : 'createChild',
                handler : this.createNewProject.createDelegate(this),
                iconCls : 'add',
                text     : '添加新项目'
            },{
                id       : 'remove',
                handler : this.removeNode.createDelegate(this),
                iconCls : 'delete',
                text     : '删除节点'
            },'-',{
                id       : 'refresh',
                handler : this.refresh.createDelegate(this),
                iconCls : 'refresh',
                text     : '刷新'
            },{
                id       : 'config',
                iconCls : 'config',
                handler : this.configInfo.createDelegate(this),
                text     : '详细配置'
            }]
        });
    },

    // 生成新项目
    createNewProject : function() {
        this.createNode(this.getRootNode());
    }
});
```

TrackerTree修改了超类中的两个函数buildToolbar()和buildContextMenu()，另外增加了一个createNewProject()函数，以这种方式将JsonTree中默认的工具条和右键菜单修改成了项目项目中左侧树形菜单中的样式。

系统右侧以JsonGrid为基础生成功能表格，还支持对某一列数据进行相关搜索，默认拥有CRUD等基本功能，进行新增和修改时会弹出对应的表单窗口，供用户填写任务信息，在进行删除操作时可以选择单条或多条记录。这些操作都可以通过简单的配置直接获取。

与TrackerTree的创建方式类似，TrackerGrid的创建代码如代码清单13-12所示。

代码清单13-12　创建TrackerGrid

```
var metaData = [
    {name: 'id',          fieldLabel: "ID",       allowBlank: true, readOnly: true},
    {name: 'projectId',   fieldLabel: "项目id",   allowBlank: false, mapping:
    'trackerProject.id', xtype: 'hidden', hideGrid: true},
    {name: 'name',        fieldLabel: "任务名称", allowBlank: false},
    {name: 'priority',    fieldLabel: "优先级",  allowBlank: false, xtype: 'priority',
    renderer: this.renderPriority},
    {name: 'severity',    fieldLabel: "严重度",  allowBlank: false, xtype: 'severity',
    renderer: this.renderSeverity},
    {name: 'status',      fieldLabel: "状态",     allowBlank: true,  xtype: 'status',
    renderer: this.renderStatus},
    {name: 'assignTo',    fieldLabel: "负责人",   allowBlank: false},
    {name: 'submitBy',    fieldLabel: "提交者",   allowBlank: false},
    {name: 'addDate',   fieldLabel: "提交时间", allowBlank: false, xtype: 'datefield'},
    {name: 'updateDate', fieldLabel: "修改时间", allowBlank: false, xtype: 'datefield'},
    {name: 'content',   fieldLabel: "内容",    allowBlank: true, xtype: 'htmleditor',
    renderer: this.renderContent},
    {name: 'file',        fieldLabel: "附件",     allowBlank: true}
];
Tracker.grid = new TrackerGrid({
    formConfig    : metaData,
    dlgWidth      : 450,
    dlgHeight     : 250,
    dialogContent : "grid_content",
    urlPagedQuery : "./trackerIssue!pagedQuery.do",
    urlSave       : "./trackerIssue!save.do",
    urlLoadData   : "./trackerIssue!loadData.do",
    urlRemove     : "./trackerIssue!remove.do",
    region: 'center',
    title: '列表'
});
```

TrackerGrid直接继承自JsonGrid。我们来看一下它所使用的初始化参数，以便对它有更进一步的了解。

- formConfig，表格中的formConfig有两种作用，它不仅可以设置新建更新时的表单格式，还可以设置表格中的ColumnModel样式，因此还可以为它设置renderer属性对某一列数据进行特殊处理。
- dlgWidth和dlgHeigth用来设置生成表单窗口的宽和高。
- urlPagedQuery，分页显示时请求的后台地址。
- urlSave，保存数据时请求的后台地址。
- urlLoadData，请求一条数据时请求的后台地址。
- urlRemove，删除记录时请求的后台地址。

在此，我们需要使用同一个formConfig参数为表格的ColumnModel和弹出窗口的表单进行设置，因此formConfig中的数据也更复杂一些。由于表格中用到了一些复杂的输入控件，比如“优先级”、“严重程度”和“状态”都需要使用特定的ComboBox。为了避免在formConfig中加入过于复杂的代码，我们选择在tracker.js中提前创建这三者的自定义类型，并为其注册对应的xtype，这样在表格里就可以直接使用xtype进行引用了。

例如，定义“优先级”的代码如代码清单13-13所示。

代码清单13-13 预先定义“优先级”子类并注册xtype

```
PriorityCombo = Ext.extend(Ext.form.ComboBox, {
    name: 'priority_name',
    hiddenName: 'priority',
    readOnly: true,
    fieldLabel: '优先级',
    valueField: 'id',
    displayField: 'name',
    typeAhead: true,
    mode: 'remote',
    triggerAction: 'all',
    emptyText: '请选择',
    selectOnFocus: true,
    allowBlank: false,
    store: new Ext.data.Store({
        proxy: new Ext.data.MemoryProxy([
            [0,'低'],
            [1,'中'],
            [2,'高']
        ]),
        reader: new Ext.data.ArrayReader({
        },['id','name'])
    })
});
Ext.reg('priority', PriorityCombo);
```

为了实现将任务绑定到对应项目中，我们对TrackerGrid进行了一些修改，使用户在新建或修改任务时自动获得左侧选中的项目，并在保存任务时将projectId作为参数传递给后台处理。

TrackerGrid也使用了Ext.extend()函数对JsonGrid进行了扩展，实现代码如代码清单13-14所示。

代码清单13-14 使用Ext.extend()扩展JsonGrid

```
TrackerGrid = Ext.extend(Ext.lingo.JsonGrid, {
    viewConfig: {
        forceFit: true
    },

    // 弹出添加对话框，添加一条新记录
    add : function() {
        TrackerGrid.superclass.add.call(this);
        this.formPanel.getForm().findField("projectId").setValue(this.projectId);
```

13

```
        },

        // 弹出修改对话框
        edit : function() {
            TrackerGrid.superclass.edit.call(this);
            this.formPanel.getForm().findField("projectId").setValue(this.projectId);
        },

        // 设置baseParams
        setBaseParams : function() {
            // 读取数据
            this.store.on('beforeload', function() {
                this.store.baseParams = {
                    filterValue: this.filter.getValue(),
                    filterTxt: this.filterTxt,
                    projectId: this.projectId
                };
            }.createDelegate(this));
        }
    });
```

代码中主要修改了add()、edit()和setBaseParams()函数，将projectId绑定到向后台提交的参数中，这样后台代码可以通过projectId判断当前任务究竟处于哪一个工程。

图13-7是在表格中新增一条任务的界面效果。



图13-7　新增一条任务

为了实现快速开发，我们不仅仅在前台对EXT进行了封装，还在后台对Struts 2以及Hibernate部分进行了对应的封装。大多数时候只需要继承超类，就可以获得表格或树形的CRUD操作。如此一来，需要编写的代码更简洁，结构更清晰，开发效率也更快了。

13.3 小结

本章详细介绍了两个有特色的实例。

VIP客户统计系统完全基于前台JavaScript开发，它演示了如何更灵活地使用`renderer()`函数渲染出更漂亮的行列效果，同时扩展了`MemoryProxy`使其实现前台分页、排序、过滤等高级功能。

Tracker任务跟踪系统更倾向于演示如何将开发中的基本功能封装为超类，这样在开发类似功能的时候只需要继承对应的超类就可以继承大多数通常都需要实现的功能。这样不止减少了代码，也使程序的结构更加清晰，极大地提高了开发效率。

第 14 章

EXT 3.x中的新特性

14

本章内容

- 介绍Ext Core
- 介绍Ext Direct
- 介绍EXT 3.0中新增的组件
- 介绍EXT 3.0中支持的Flash报表
- EXT 3.1带来的新特性
- EXT 3.2带来的新特性

2009年7月6日，发布了EXT 3.0正式版，我们终于迎来了2.x到3.x这一重大版本升级。虽然之前广为宣传的Web Designer还未正式亮相，但3.0也为我们提供了多项重要的功能改进。EXT团队不仅为我们提供了更多、更绚丽的控件，在基础功能和与后台服务的数据传输方面也下了不小的功夫。下面我们来逐一介绍EXT 3.x中新增加的特性。

14.1 介绍 Ext Core

EXT 3.0大大增强了自身的模块化管理机制，正因如此，EXT官方已经将包含核心功能的Ext Core抽离出来，作为单独的发布包提供给我们下载。在Ext Core中包含的都是Web开发所需的基本操作，其中并不包括诸如表格、菜单等组件，我们可以将Ext Core看做是与prototypejs、jquery或是Mootools同一级别的，用于底层JavaScript开发的核心工具库。Ext Core告诉我们EXT并非只能用于企业开发，它其中蕴含的基础功能完全可以适用于网络开发。

Ext Core中包含的功能有：DOM遍历与操作、Ajax、事件处理、动画、模板和面向对象机制。从源代码结构可以看到整个Ext Core分为4大部分。

14.1.1 adapter

在adapter目录包含7个文件。

- ext-base-ajax.js，实现了`Ext.lib.Ajax`类的功能，这是Ext中最基本的Ajax底层实现。
- ext-base-anim.js，实现了`Ext.lib`中基本动画功能。
- ext-base-anim-extra.js，对 `Ext.lib`中的动画功能进行扩展，实现了平滑变换与颜色变换的功能。
- ext-base-begin.js，只包含`Ext.fly`的定义。

- ext-base-dom.js，实现了对DOM的大小位置等属性的操作。
- ext-base-end.js，对IE进行特别处理，在触发页面unload事件时取消document对象上定义的事件绑定。
- ext-base-event.js，对浏览器中原生事件进行了封装，返回统一的处理类。

14.1.2 core

在core目录包含14个文件。

- CompositeElementLite.js，即轻量级的复合元素。
- DomHelper.js，提供对DOM的各种辅助操作功能。
- DomQuery.js，提供了对DOM的各种查询操作。
- Element.fx.js，提供了与Element对象相关的动画效果。
- Element.insertion.js，提供了对元素的插入替换等功能。
- Element.js，定义了应用最广的Element类，几乎所有的操作都可以通过封装的Element来实现。
- Element.position.js，提供了对Element位置操作的功能。
- Element.scroll.js，提供了对Element滚动的支持。
- Element.style.js，提供了Element样式的支持。
- Element.traversal.js，提供了对Element的遍历操作，注入findParent()、findParentNode()等。
- EventManager.js，定义了事件管理器，支持对docReady等事件的特殊处理。
- Ext.js，初始化了Ext这个定义命名空间，并提供了一系列判断当前使用的浏览器类别的静态属性。
- Fx.js，实现了所有动画效果，所有动画效果可以直接绑定在Element中使用。
- Template.js，提供了对模板的支持。

14.1.3 data

在data目录包含1个文件。

- Connection.js，在Ext.lib.Ajax这个底层的基础之上实现了统一的Ext.Ajax静态类，其中支持更易用的Ajax操作。

14.1.4 util

在util目录包含4个文件。

- DelayedTask.js，提供简单的定时任务操作。
- JSON.js，提供对JSON操作的支持，其中的方法可以直接通过Ext.encode()和Ext.decode()调用。
- Observable.js，实现了观察者设计模式，所有支持自定义事件的组件都可以直接继承这个

类，从而获得所有事件功能。

- TaskMgr.js，定义了事件管理器，可以对DelayTask进行统一管理。

14.1.5　扩展实例

Ext Core的官方发布包还提供了几个基于Ext Core的扩展组件，因为Ext Core中只包含了底层核心库，并没有包含任何功能复杂的组件。下面就来看一下基于这个简易版的Ext Core核心库可以实现何种效果。

假若我们希望在页面上实现类似TabPanel的功能，基于Ext Core只需要编写很少几行代码就可以实现，如下所示：

```
Ext.onReady(function(){
   Ext.select('.tab-buttons-panel').on('click', function(e, t) {
       Ext.get(t).radioClass('tab-show');
       Ext.get('content' + t.id.slice(-1)).radioClass('tab-content-show');
   }, null, {delegate: 'li'});
});
```

这段代码的功能是在页面加载完成之后，使用Ext.select()监听CSS的class值为tab-buttons-panel的元素，并监听元素中由li元素产生的click事件，在用户使用鼠标点击这些元素时就会触发相应的函数进行处理。

当用户使用鼠标点击这些元素时，首先使用Ext.fly()获得Element，然后调用radioClass()，这样就会为当前的元素批量实现切换CSS样式的效果。它可以保证一批元素中仅有当前被点击的元素会被设置上tab-show这个样式，这样就可以看到当前元素也就是标签头部是处于被选中的状态。

然后使用Ext.get()函数，通过点击的元素，也就是标签头部的id获得对应标签内容的元素，也通过radioClass()函数批量切换CSS样式，这样操作的结果是被点击标签头部的内容部分会显示出来。

为便于大家理解，下面给出了对应的HTML代码内容：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html lang="en">
<head>
    <title>Ext Tabs Example</title>

    <link href="../examples.css" rel="stylesheet" />
    <link href="tabs.css" rel="stylesheet" />

    <script src="../../ext-core-debug.js"></script>
    <script src="tabs.js"></script>
</head>
<body>
    <div class="tab_container">
        <div class="tab-buttons-panel">
            <ul>
                <li id="tab1" class="tab-show">
                    <span>Nature</span>
```

```
            </li>
            <li id="tab2">
                <span>Vehicles</span>
            </li>
            <li id="tab3">
                <span>Animals</span>
            </li>
        </ul>
    </div>
    <div id="content1" class="tab-content tab-content-show">
        <div class="tab-content-panel-border">
            <div class="tab-content-panel">
                <b>Things in Nature</b>
                <ul>
                    <li>Plants</li>
                    <li>Ocean</li>
                    <li>Trees</li>
                    <li>Skies</li>
                    <li>Mountains</li>
                </ul>
            </div>
        </div>
    </div>
    <div id="content2" class="tab-content">
        <div class="tab-content-panel-border">
            <div class="tab-content-panel">
                <b>Types of Vehicles</b>
                <ul>
                    <li>Cars</li>
                    <li>Boats</li>
                    <li>Trucks</li>
                    <li>Bikes</li>
                </ul>
            </div>
        </div>
    </div>
    <div id="content3" class="tab-content">
        <div class="tab-content-panel-border">
            <div class="tab-content-panel">
                <b>Types of Animals</b>
                <ul>
                    <li>Tigers</li>
                    <li>Elephants</li>
                    <li>Fish</li>
                    <li>Birds</li>
                </ul>
            </div>
        </div>
    </div>
</div>
</body>
</html>
```


从以上代码中可以看出，表示标签面板头部的div的class值为tab-buttons-panel，它下

面有3个li元素，每个li元素将显示为一个标签头部，每个标签头部都与下面部分的div一一对应，可以看到它们之间id的对应关系，这些div包含的就是标签内容。

点击任意一个li元素时，就会触发click事件，事件监听函数会根据点击的标签头部批量切换CSS样式，从而达到显示不同标签内容的效果，最终显示效果如图14-1所示。

也许有人会嫌这里的显示效果太简陋了，比不上之前见过的EXT所展示出来的绚丽效果，这是因为Ext Core并没有包含任何与页面显示相关联的内容，没有提供大量的图片背景，也没有提供丰富的样式设置。从Web开发的角度来讲，这解放了前端开发人员。因为之前EXT提供的默认主题过于复杂，所以有人曾提出了美工无用武之地的窘境。现在Ext Core像其他底层JavaScript开发库一样，只关注基本操作，将布局和显示样式设置的门槛降低，我们就可以使用它结合自定义的样式设计实现自己所需的功能了。

下面再来看一个基于Ext Core实现的lightbox效果，如图14-2所示。

Nature　Vehicles　Animals

Things in Nature

- Plants
- Ocean
- Trees
- Skies
- Mountains

图14-1　基于Ext Core实现TabPanel

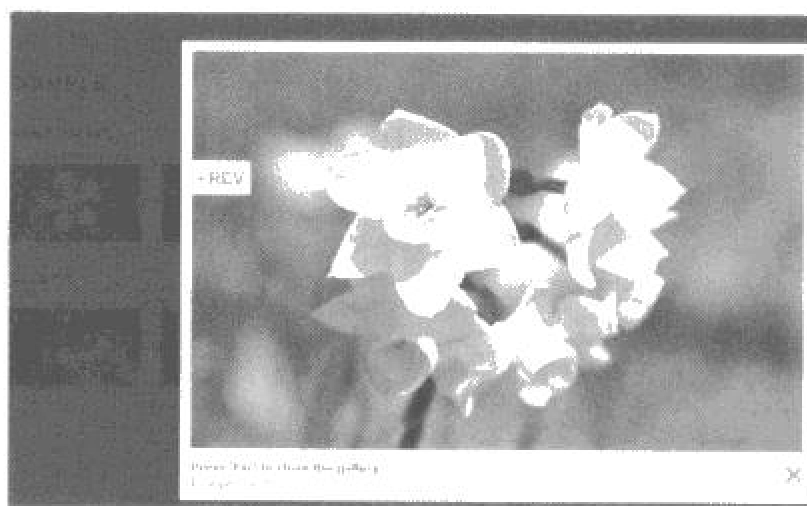


图14-2　基于Ext Core开发的lightbox组件

官方发布包中还包括了更多基于Ext Core扩展的组件，可以为我们提供更多可以直接利用的功能。实例以及代码可以在发布包的example目录下找到。

14.2　介绍 Ext Direct

Ext Direct是一套EXT 3.0中出现的数据交换API，它的作用是屏蔽服务器后端平台的差异，使用同一套API实现前台应用与后台服务之间的数据交互。

对Java用户来说，Ext Direct似乎就是另一个DWR，实际上两者的工作原理也是基本相同，都是根据后台对应的功能方法生成前台中对应的JavaScript函数。在调用JavaScript函数时，会自动发起Ajax请求调用后台提供的功能，并自动处理响应的数据。只是DWR仅为Java量身打造，无法使用在其他平台上，而Ext Direct为各个主流平台都提供了实现，而前台生成的JavaScript脚本也是完全基于EXT中的实现，更易于使用在基于EXT的项目中。

14.2.1　Ext Direct

使用Ext Direct的第一步是为后台暴露的远程方法声明API，一个实际的api.js代码内容如下：

```
Ext.app.REMOTING_API = {
    type: 'remoting',
    url: 'jsp/router.jsp',
    actions: {
        TestAction: [{
            name: 'doEcho',
            len: 1
        }, {
            name: 'multiply',
            len: 1
        }, {
            name: 'getTree',
            len: 1
        }],
        Profile: [{
            name: 'getBasicInfo',
            len: 1
        }, {
            name: 'getPhoneInfo',
            len: 1
        }, {
            name: 'getLocationInfo',
            len: 1
        }]
    }
};
```

上面代码中定义了类型为远程调用（remoting）的API，对应的后台url是jsp/router.jsp，这样在它内部所定义的actions调用时都会向这里定义的url（jsp/router.jsp）发送请求。

代码的actions属性中定义了所暴露的类名和类下面包含的方法定义，这里定义了两个类TestAction和Profile。其中TestAction下包含3个方法doEcho()、multiply()和getTree()，这3个方法在调用时都需要传入一个参数。下面定义的Profile与TestAction基本类似，它也包含3个方法getBasicInfo()、getPhoneInfo()和getLocationInfo()，这3个方法在调用时也都需要传递一个参数。

在使用Ext Direct之前，我们就需要把这个api.js导入到页面中：

```
<script type="text/javascript" src="api.js"></script>
```

之后还要在JavaScript中将api.js中定义的Ext.app.REMOTING_API设置到Ext Direct中，Ext Direct会根据其中的配置，自动生成对应的对象。设置代码如下所示：

```
Ext.Direct.addProvider(
    Ext.app.REMOTING_API,
    {
        type:'polling',
        url: 'jsp/poll.jsp'
    }
);
```

14

实际上，我们可以同时配置多个Ext Direct的远程方法，上面的代码中不仅设置了api.js中配置的Ext.app.REMOTING_API，还附加了额外的一段类型为polling的配置。有关Ext Direct

支持的调用类型将在后面详细介绍。

在完成了Ext Direct的设置之后，就可以在代码中直接调用这些功能函数了。如果之前使用过DWR的话，应该对调用的方式比较熟悉，调用远程函数的代码如下所示：

```
TestAction.doEcho(text.getValue(), function(result, e){
  var t = e.getTransaction();
  out.append(String.format('<p><b>Successful call to {0}.{1} with response:</b><xmp>
  {2}</xmp></p>',
    t.action, t.method, Ext.encode(result)));
  out.el.scrollTo('t', 100000, true);
});
```

因为在api.js中声明了TestAction类，并定义了TestAction类中的doEcho()方法，因此这里可以直接使用TestAction.doEcho()进行调用，这样就会直接调用到后台的TestAction类的doEcho()方法，并将返回的结果以回调函数形式传递给前端的JavaScript脚本。

在回调函数中，result包含了返回的所有数据，回调函数的第二个参数e中包含了此次调用的各种设置，最主要的就是事务t，事务包含了此次调用的actoin、method等参数。这些都是交由Ext Direct自动处理的，前台与后台交互的过程以及实际中数据处理的方式，对最终用户来说都是没有必要了解的，只需在前端的JavaScript中调用对应的函数，就会从后台获得相应的数据。

Ext Direct主要包含两种调用方式，一种是我们上面提到的远程调用方式，它需要通过api.js来映射后端暴露的服务方法，通过对应的Provider将API配置转换为前端的功能对象，交由用户使用。另一种调用方式为轮询，它的作用是实现客户端轮询。我们只需要为轮询类型的Ext Direct指定对应的url，EXT就会自动在后台启动轮询任务，每过一段时间自动向后台发起请求，获得对应的结果。在轮询操作完成后会以事件提醒的方式交由前端的JavaScript代码处理，因此在将Ext Direct配置为轮询方式之后，还需要在代码中设置对应的事件监听器，通过回调函数获得轮询请求的结果，对应代码如下所示：

```
Ext.Direct.on('message', function(e){
    out.append(String.format('<p><i>{0}</i></p>', e.data));
            out.el.scrollTo('t', 100000, true);
});
```

上述代码中，我们设置了对message事件的监听器。在轮询发出message事件时，就会自动触发回调函数，这里就会对out这个元素的内容进行更新，并进行自动滚动。

EXT 3.0中基于Ext Direct提供了为数不少的底层支持，我们可以在表格、树形、表单中选择使用Ext Direct方式与后台进行数据交互。

14.2.2 洞悉 Ext Direct 的原理

Ext Direct到底是怎样实现在前端JavaScript中直接调用后台服务所暴露的方法呢？我们可以猜到，中间肯定是借助了Ajax进行数据交互，但这些数据的形式到底是怎么样的呢？如果我们确实打算选择使用Ext Direct作为连接前后端应用的桥梁时，是否可以保证这些封装后的操作可以完全在我们的掌握之中呢？

答案是肯定的，Ext Direct实际上只利用JSON作为数据的交换格式，在原来的Ajax基础上提

供了自身的数据交互协议，应用的前后端使用同种协议进行调用，从而达到屏蔽后端实现的目的。

当调用Ext Direct的函数时，EXT会根据API配置发送图14-3中格式的请求。

```
POST http://localhost:8080/direct/jsp/router.jsp  200 OK 749ms
Headers  Post  响应
{"action":"TestAction","method":"doEcho","data":["1"],"type":"rpc","tid":2}
```

图14-3 Ext Direct触发的请求

- action表示期望调用的远程类。
- method表示期望调用的远程方法。
- data表示调用远程方法时传递的参数，它的类型是一个数组，数组中的参数个数与Ext.app.REMOTE_API中对应的len值相同。
- type表示使用rpc方式进行远程调用。
- tid表示此次调用的事务id，它可以用来在批量调用中区分每个对应的请求和响应。

后台返回的响应信息内容如图14-4所示。

```
POST http://localhost:8080/direct/jsp/router.jsp  200 OK 749ms
Headers  Post  响应

{
    type: 'rpc',
    action: 'TestAction',
    method: 'doEcho',
    tid: 2,
    result: 'x'
}
```

图14-4 处理Ext Direct请求后返回的信息

- result表示远程方法调用后返回的内容。
- action表示调用的远程类。
- method表示调用的远程方法。
- type表示使用rpc方式进行远程调用。
- tid表示此次调用的事务id，它可以用来在批量调用中区分每个对应的请求和响应。

从这两个对应的请求与响应内容可以看到，所有的action、method、type、tid都是一一对应的。实际上，Ext Direct会根据这些对应的参数判断对应的操作该发送给后台的哪个方法处理，并且后台返回的响应信息应该交给哪一个回调函数。

这就是Ext Direct的基础实现，其上层所支持的一切功能都由这样的请求与响应数据格式支撑，而不同平台实现的Ext Direct后端，都需要解析请求中特定的信息内容，根据其中定义的参数在后端进行调用，并将返回的结果组装成Ext Direct所需的格式返回给前端，这样实现的循环确保了Ext Direct的正常应用。

14.2.3 使用 directjngine 支持 Ext Direct

EXT官方并未提供Java平台下的Ext Direct支持组件，但是Java开源社区的力量实在太庞大了，第三方已经推出了好几套在Java平台下支持Ext Direct的组件库。其中比较成熟的是directjngine，它支持直接在POJO中使用注解配置Ext Direct调用的远程方法。下面我们就来看一下如何使用directjngine支持Ext Direct。

directjngine的官方网站在http://code.google.com/p/directjngine/，最新的发布版本是directjngine-1.0.zip，可以在http://code.google.com/p/directjngine/downloads/list下载到发布包和源代码。需要注意的是directjngine中使用了大量JDK 5中添加的特性，因此只能使用在JDK 5及以上版本。

directjngine的官方发布包中已经包含了运行时必须的第三方依赖库，可以在lib目录下找到这些依赖库，如下所述。

- directjngine.1.0.jar：directjngine核心库。
- gson-1.3.jar：JSON与Java对象实现相互转换的工具库。
- log4j-1.2.15.jar：日志库。
- commons-fileupload-1.2.1.jar和commons-io-1.4.jar：用来支持文件上传。
- jargs-1.0.jar、rhino-1.6R7.jar和yuicompressor-2.4.2.jar：用来生成JavaScript脚本。

下一步我们需要编写提供给前台调用的远程方法，这里借用EXT官方例子中提供的Ext Direct实例，它们分别是`TestAction.java`和`Profile.java`。

`TestAction.java`主要用来演示使用远程方法为Ext Direct提供数据，其中定义了3个方法，分别是`doEche()`、`multiply()`和`getTree()`，具体代码内容如下所示：

```
package sample;

import java.util.ArrayList;
import java.util.List;

import com.softwarementors.extjs.djn.config.annotations.DirectMethod;

public class TestAction {
  @DirectMethod
  public String doEcho( String data ) {
    return data;
  }

  @DirectMethod
  public double multiply( String num ) {
    double num_ = Double.parseDouble(num);
    return num_ * 8.0;
  }

  public static class Node {
    public String id;
    public String text;
    public boolean leaf;
  }
```

```
    @DirectMethod
    public List<Node> getTree( String id ) {
      List<Node> result = new ArrayList<Node>();
      if( id.equals("root")) {
        for( int i = 1; i <= 5; ++i ) {
          Node node = new Node();
          node.id = "n" + i;
          node.text = "Node " + i;
          node.leaf = false;
          result.add(node);
        }
      }
      else if( id.length() == 2) {
        String num = id.substring(1);
        for( int i = 1; i <= 5; ++i ) {
          Node node = new Node();
          node.id = "id" + i;
          node.text = "Node " + num + "." + i;
          node.leaf = true;
          result.add(node);
        }
      }
      return result;
    }
  }
```

从上述的TestAction.java中可以看到，代码内容与我们通常使用的POJO基本没有什么区别，唯一特别的就是方法上使用了directjngine提供的注解。实际上directjngine就是通过这些@DirectMethod注解与Ext Direct中的请求与响应进行关联，实现远程方法调用功能的。

下面我们还需要在web.xml中为directjngine进行相应的配置，这样它才可以处理客户端发送来的请求，并返回相应的响应数据，代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app_2_4.xsd"
          version="2.4">

  <!-- DirectJNgine servlet -->
  <servlet>
    <servlet-name>DjnServlet</servlet-name>

    <servlet-class>com.softwarementors.extjs.djn.servlet.DirectJNgineServlet</servlet-class>

    <init-param>
      <param-name>debug</param-name>
      <param-value>true</param-value>
    </init-param>

    <init-param>
      <param-name>minify</param-name>
      <param-value>true</param-value>
```

14

```
        </init-param>

        <init-param>
          <param-name>providersUrl</param-name>
          <param-value>djn/directprovider</param-value>
        </init-param>

        <init-param>
          <param-name>batchRequestsMultithreadingEnabled</param-name>
          <param-value>true</param-value>
        </init-param>

        <init-param>
          <param-name>apis</param-name>
          <param-value>
            sample
          </param-value>
        </init-param>

        <init-param>
          <param-name>sample.apiFile</param-name>
          <param-value>sample/api.js</param-value>
        </init-param>

        <init-param>
          <param-name>sample.apiNamespace</param-name>
          <param-value>Ext.app</param-value>
        </init-param>

        <init-param>
          <param-name>sample.classes</param-name>
          <param-value>
          sample.TestAction
          </param-value>
        </init-param>

        <load-on-startup>1</load-on-startup>
      </servlet>

      <servlet-mapping>
        <servlet-name>DjnServlet</servlet-name>
        <url-pattern>/djn/directprovider/*</url-pattern>
      </servlet-mapping>
    </web-app>
```

从web.xml中的配置可以看到，directjngine将用来处理以/djn/directprovider/为前缀的所有请求。如果前台发送的请求是以/djn/directprovider/开始的，那么这个请求就会自动分配给com.softwarementors.extjs.djn.servlet.DirectJNgineServlet进行处理。

之后就可以在客户端导入directjngine自动发布的api.js使用Ext Direct了。

14.3 介绍 EXT 3.0 中新增的组件

EXT 3.0中又新增了几种拥有华丽外观的组件，下面我们就来逐一介绍这些新组件。

14.3.1 行编辑器

如图14-5所示，最新添加到表格中的行编辑器（Row Editor）是一个令人惊叹的组件，我们借此可以一次性为某一行的数据进行批量修改，不必再像EditorGridPanel中一样为了修改每一个单元格进行多余的双击了。而且行编辑器还支持保存（save）和取消（cancel）操作，如此一来就算是编辑错误，也可以取消掉，不会对实际数据产生影响。

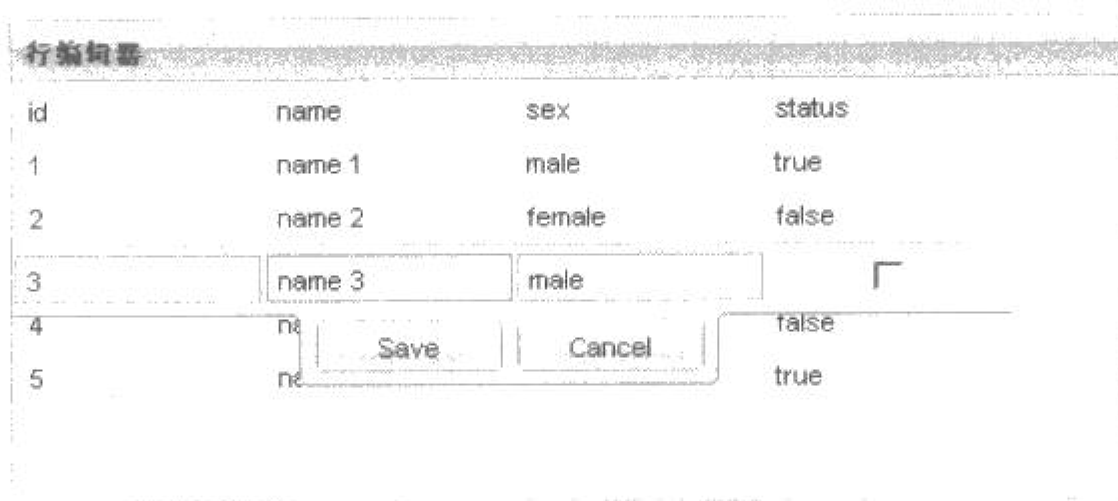


图14-5 行编辑器的编辑状态

作为一个额外提供的插件，行编辑器使用起来尤其方便。要启用行编辑器，我们只需在表格的plugins属性中添加一个Ext.ux.RowEditor的实例，然后在双击表格中某一行的时候就会看到行编辑器。

```
plugins: [new Ext.ux.grid.RowEditor()],
```

不过为了定义每一列单元格使用的输入控件，我们还要为columns添加一个editor属性，里边定义了输入控件的xtype和校验规则。

```
editor: {
    xtype: 'textfield',
    allowBlank: false
}
```

这里我们可以定义任意一个表单输入控件，比如checkbox：

```
editor: {
    xtype: 'checkbox'
}
```

最后还要记得，因为这个扩展不属于EXT核心包，所以在使用时需要先把examples/grid中的RowEditor.js引入页面中，还要添加额外的CSS样式和对应图片。切记切记！

14.3.2 进度条分页组件

这个组件的功能是把原来的分页工具条变成进度条的样子，效果绚丽了好几倍，如图14-6所示。

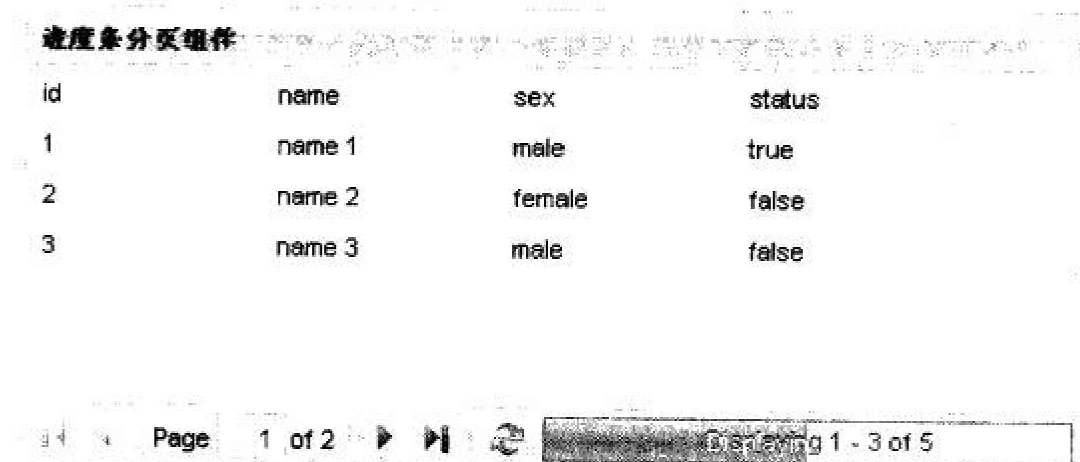


图14-6 用进度条的方式来显示分页

使用方法尤为简单，只要在Ext.PagingToolbar中添加Ext.ux.ProgressBarPager这个组件就可以了。当然还要记得在使用前在页面中引入ProgressBarPager.js这个文件。

示例代码如下，个人感觉使用这个组件控制分页也不精准，但是从演示的方面来说，还是非常有用的。

```
bbar: new Ext.PagingToolbar({
    pageSize: 3,
    store: store,
    displayInfo: true,
    plugins: new Ext.ux.ProgressBarPager()
}),
```

14.3.3 缓冲式表格视图

这个扩展的理念应该是来自extjs.com官网上的LiveGrid组件，因为一般认为EXT的表格在显示超过200条数据的时候就会出现反应迟缓的现象，虽然不知道为什么非要在一个页面上显示这么多数据。不过确实有认为EXT存在性能问题的开发人员说，他们每次都在EXT的表格里显示1 000条数据，这是由他们的技术总监决定的，而且认为EXT不支持除此之外的显示方式，因此就来向我们抱怨EXT的性能问题。

不过有的时候确实有显示数据过多反应速度缓慢的情况出现，这时就要借助3D游戏里的一种实时渲染的概念了，即表格里显示哪些数据，就把哪些数据画出来，这样不会去处理不显示的那些DOM，也就提高了整体的效率。

我们分别用普通表格视图（GridView）和缓冲式表格视图（BufferView）显示1 000条数据进行实验，这会在初始化的时候明显感觉到速度的不同，但在拖曳滚动条的时候没感觉到差多少。

使用缓冲式表格视图是非常简单的，只需单独为表格设置一个view参数就可以了。

```
view: new Ext.ux.BufferView({
    rowHeight: 20,
    scrollDelay: false
})
```

14.3.4 标签面板的滚动菜单

这是一个用于TabPanel的组件，可以在标签过多的时候显示一个下拉菜单，供用户进行选择，如图14-7所示。

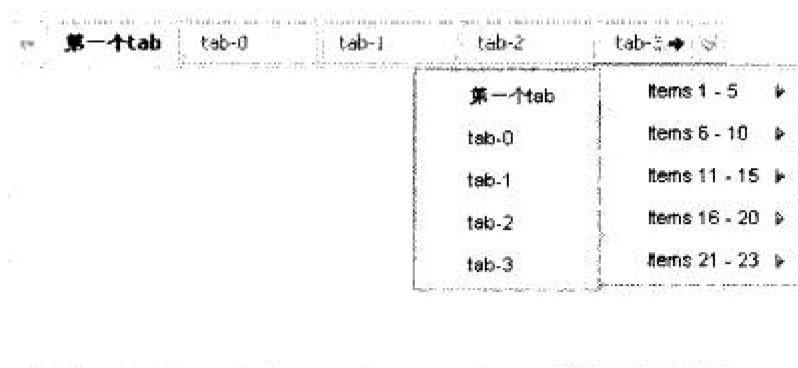


图14-7 使用滚动菜单显示溢出的TabPanel

因为又是一个扩展，我们需要做的只是设置plugins，不过还要记得引入tabs下的TabScrollerMenu.js和TabScrollerMenu.css，一定要记得在页面中导入对应的CSS样式，否则标签多出来的时候页面就会出错。

```
var scrollerMenu = new Ext.ux.TabScrollerMenu({
    maxText  : 15,
    pageSize : 5
});

var tabs = new Ext.TabPanel({
    height: 200,
    width: 400,
    activeTab: 0,
    enableTabScroll: true,
    resizeTabs: true,
    minTabWidth: 75,
    frame: true,
    plugins: [scrollerMenu],
    items: [{
        title : '第一个tab'
    }],
    renderTo: 'tabs'
});
```

14.3.5 处理工具条溢出

当工具条中按钮过多时，EXT会自动隐藏右侧过多的按钮，并将这些按钮以下拉菜单的形式显示在工具条右侧。如图14-8所示。

这个功能十分贴心，不需要设置任何插件，直接集成在Ext.Toolbar中。在工具条中的项目很多的时候自动使用这个功能，不用我们多写一行代码。

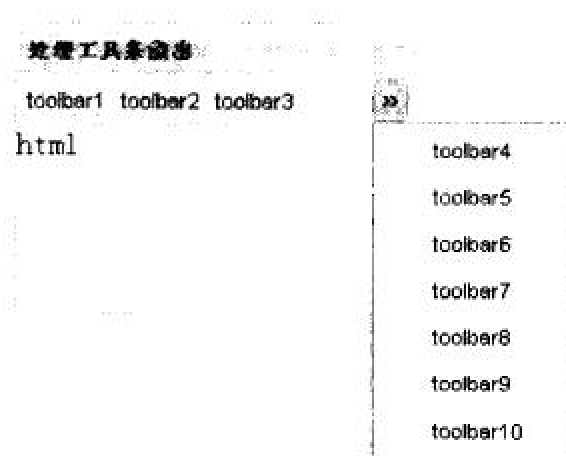


图14-8 对工具条溢出的处理

14.3.6　列表视图

EXT 3.x中新增的列表视图（ListView）组件是一种整合型的列表视图，它比表格更简单、更轻巧、功能更少，也能满足大部分的显示功能。它也是因为好多人在抱怨表格中包含的功能太多，希望EXT可以提供一套可以实现简单显示效果的组件而出现的。

列表视图组件如图14-9所示。

File	Last Modified	Size
name	04-01 10:10 am	1000 bytes
name	04-01 10:10 am	1000 bytes

图14-9　使用列表视图实现轻量级表格

图14-9中的实现代码如下所示。

```
var listView = new Ext.ListView({
    store: new Ext.data.SimpleStore({
        data: [
            ['name', '2009-04-01 10:10:10', 1000],
            ['name', '2009-04-01 10:10:10', 1000]
        ],
        fields: ['name', {
            name: 'lastmod',
            type: 'date',
            dateFormat: 'Y-m-d H:i:s'
        }, 'size']
    }),
    multiSelect: true,
    emptyText: 'No data to display',
    reserveScrollOffset: true,

    columns: [{
        header: 'File',
        dataIndex: 'name'
    },{
        header: 'Last Modified',
        dataIndex: 'lastmod',
        tpl: '{lastmod:date("m-d h:i a")}'
    },{
        header: 'Size',
        dataIndex: 'size',
        tpl: '{size:fileSize}',
        align: 'right'
    }]
});
var panel = new Ext.Panel({
    title: '列表视图',
    layout: 'fit',
    width: 300,
    height: 200,
    items: [listView],
    renderTo: 'listview'
});
```

因为ListView不是Panel的子类，如果想设置标题、布局等，就要把ListView再放到Panel中。

14.3.7 工具条中的分组按钮

这个组件是在EXT 3.0高级按钮基础上实现的扩展。虽然在它自带的例子里，那些放在工具条上的按钮都很酷，但是也给人很烦乱的感觉，实际开发中如何使用这个新增功能，还是看各位的功底了。

工具条按钮如图14-10所示。

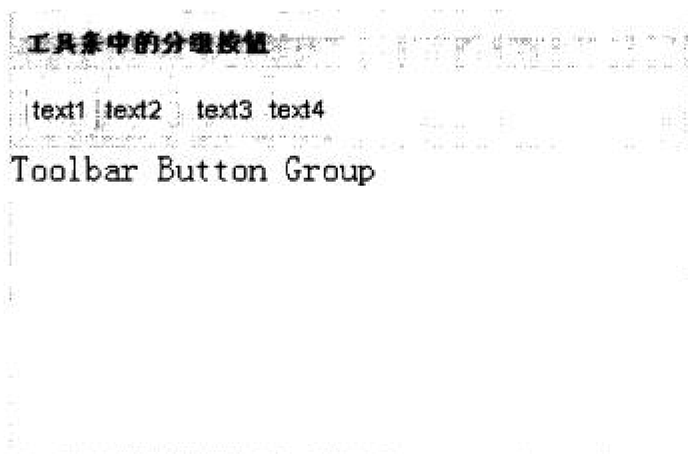


图14-10 工具条中使用分组按钮

首先需要设置的是xtype：'buttongroup'，然后在其中添加各式各样的按钮，这些按钮的详细用法参考第9章。示例代码如下所示。

```
tbar: new Ext.Toolbar([{
    xtype: 'buttongroup',
    items: [{
        text: 'text1'
    }, {
        text: 'text2'
    }]
}, {
    xtype: 'buttongroup',
    items: [{
        text: 'text3'
    }, {
        text: 'text4'
    }]
}]),
```

14.3.8 高级按钮

EXT 3.x提供的按钮太复杂了，我们面对的不再是以前那种要么是图片要么是文字的简陋选择。如今支持的是图文混排和各式各样的对齐格式，如果真要一一讲解，就会远远超出现有的篇幅。所以，到底EXT 3.x中支持了多少种按钮效果，还是靠大家慢慢去examples里体味，这里只给出几个基本例子，让大家饱饱眼福。抛砖引玉哦。如图14-11所示。



图14-11 实现多种样式的按钮

如今按钮最大的改进就是支持了3种规格：small、medium和large。这样我们可以选择在各种规格的按钮上增加文字或者是图形，获得多种多样的按钮效果。

```
var panel = new Ext.Panel({
    title: '高级按钮',
    width: 300,
    height: 200,
```

```
        defaultType: 'button',
        items: [{
            text: 'Small',
            scale: 'small'
        }, {
            text: 'Medium',
            scale: 'medium'
        }, {
            text: 'Large',
            scale: 'large'
        }],
        renderTo: 'buttons'
    });
```

14.3.9　竖直分组的标签面板

以前总有人说想要竖直的标签面板，现在这个功能来了，如图14-12所示。

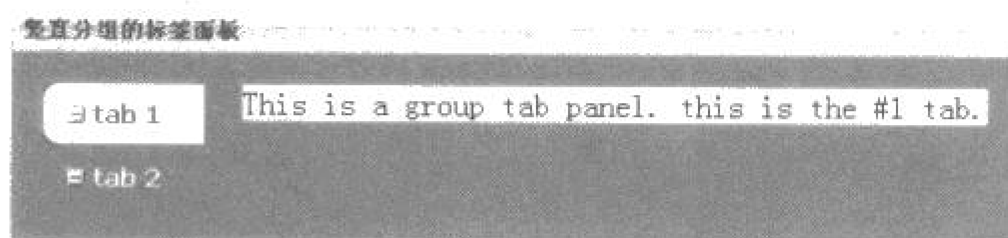


图14-12　竖直分组的标签面板

为了使用竖直标签面板，我们要为页面引入3个文件，分别是GroupTabPanel.js、GroupTab.js和一个样式文件grouptabs.css。

其实竖直标签面板也继承自我们以前经常使用的TabPanel，只是这次它的标签可以放在左侧了。实际使用的时候首先需要配置最外层组件的xtype，这里要使用xtype: 'grouptabpanel'，然后在竖直分组的标签面板（grouptabpanel）里设置activeGroup，这是告诉EXT默认显示哪个分组。实际上每个grouptabpanel的items都是一个分组，每个分组里又有多个面板，这样最后组成了一份复杂的层级结构，竖直分组的标签面板中使用activeGroup设置默认显示哪个组，而组中又使用mainItem设置默认显示哪个面板。实现代码如下所示。

```
var panel = new Ext.Panel({
    title: '竖直分组的标签面板',
    width: 500,
    height: 200,
    items: [{
        xtype: 'grouptabpanel',
        tabWidth: 80,
        activeGroup: 0,
        items: [{
            mainItem: 0,
```

```
                title: '001',
                items: [{
                    title: 'tab 1',
                    tabTip: 'Tickets tabtip',
                    html: 'This is a group tab panel. this is the #1 tab.'
                }]
            }, {
                mainItem: 0,
                title: '002',
                items: [{
                    title: 'tab 2',
                    tabTip: 'Tickets tabtip',
                    html: 'This is a group tab panel. this is the #2 tab.'
                }]
            }]
        }],
        renderTo: 'panel'
    });
```

14.4 在 EXT 3.0 中使用 Flash 报表

EXT 3.0使用的Flash Chart来源于YUI，其中提供了包括柱状图、饼状图等多种图形报表，并提供了图标与EXT组件之间的完美整合，不仅可以直接在Ext.Panel中显示图表，还可以通过Ext.data.Store为图表提供数据。

首先定义一个JsonStore为我们现在所要演示的所有图表提供数据，代码如下：

```
var store = new Ext.data.JsonStore({
    fields:['name', 'visits', 'views'],
    data: [
        {name:'Jul 07', visits: 245000, views: 3000000},
        {name:'Aug 07', visits: 240000, views: 3500000},
        {name:'Sep 07', visits: 355000, views: 4000000},
        {name:'Oct 07', visits: 375000, views: 4200000},
        {name:'Nov 07', visits: 490000, views: 4500000},
        {name:'Dec 07', visits: 495000, views: 5800000},
        {name:'Jan 08', visits: 520000, views: 6000000},
        {name:'Feb 08', visits: 620000, views: 7500000}
    ]
});
```

JsonStore中包含3列数据，分别是name、visits和views，其中name的类型是字符串，其他两列的数据类型为数字。下面所演示的所有图表中显示的数据都来自于这个JsonStore。

14.4.1 柱状图

实现柱状图（ColumnChart）的代码如下所示：

```
new Ext.Panel({
    title: 'Chart',
    renderTo: 'container',
    width:500,
```

```
        height:300,
        layout:'fit',

        items: {
            xtype: 'columnchart',
            url: '../../resources/charts.swf',
            store: store,
            xField: 'name',
            yField: 'visits',
            listeners: {
                itemclick: function(o){
                    var rec = store.getAt(o.index);
                    Ext.example.msg('Item Selected', 'You chose {0}.', rec.get('name'));
                }
            }
        }
    });
```

将xtype指定为columnchart，即在一个Panel中创建了柱状图，图表的数据来自于上面定义好的store，xField指定横轴数据显示name列的数据，yField指定纵轴数据显示vistis列的数据，最终的显示效果如图14-13所示。

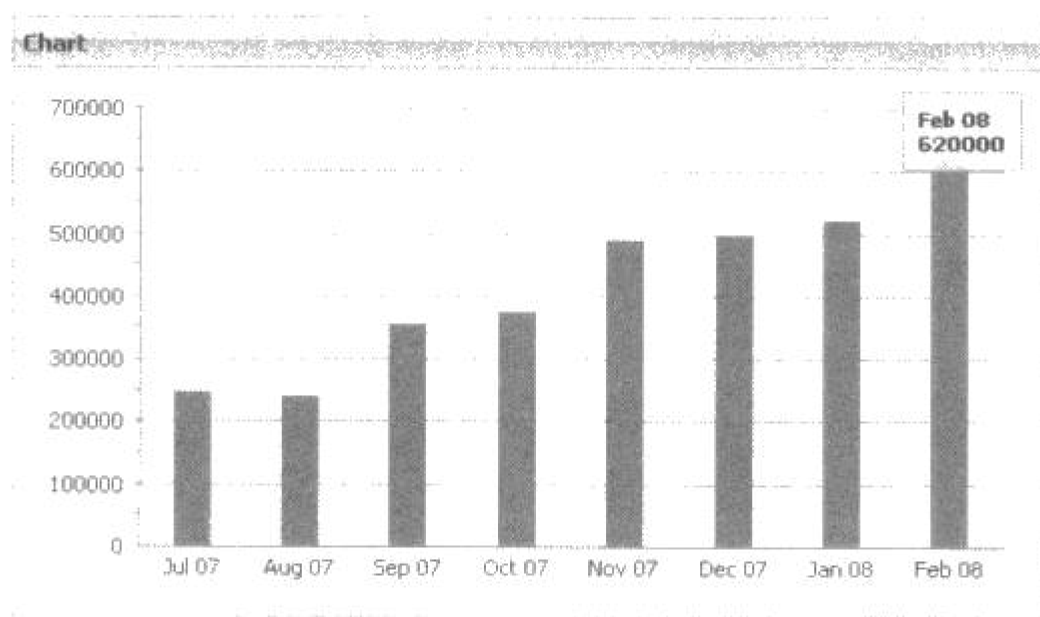


图14-13　柱状图

14.4.2　横向柱状图

实现横向柱状图（BarChart）的代码如下所示：

```
new Ext.Panel({
    title: 'Chart',
    renderTo: 'container',
    width:500,
    height:300,
    layout:'fit',
```

```
        items: {
            xtype: 'barchart',
            url: '../../resources/charts.swf',
            store: store,
            yField: 'name',
            xField: 'visits',
            listeners: {
                itemclick: function(o){
                    var rec = store.getAt(o.index);
                    Ext.example.msg('Item Selected', 'You chose {0}.', rec.get('name'));
                }
            }
        }
    });
```

将xtype指定为barchart，即在一个Panel中创建了横向柱状图，图表的数据来自于上面定义好的store，xField指定横轴数据显示visits列的数据，yField指定纵轴数据显示name列的数据，最终的显示效果如图14-14所示。

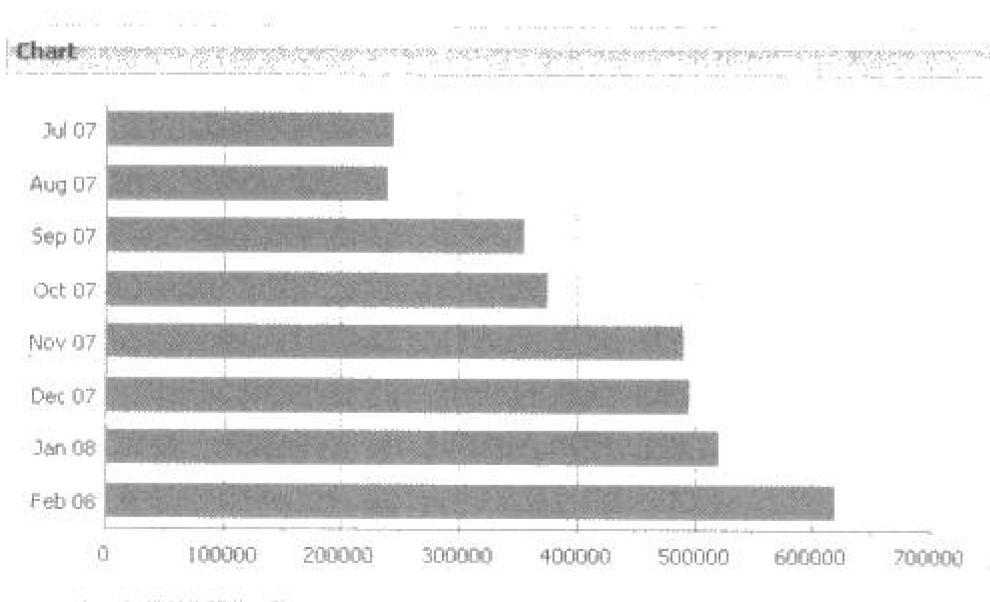


图14-14 横向柱状图

14.4.3 折线图

实现折线图（LineChart）的代码如下所示：

```
new Ext.Panel({
    title: 'Chart',
    renderTo: 'container',
    width:500,
    height:300,
    layout:'fit',

    items: {
        xtype: 'linechart',
        url: '../../resources/charts.swf',
```

14

```
            store: store,
            xField: 'name',
            yField: 'visits',
            listeners: {
                itemclick: function(o){
                    var rec = store.getAt(o.index);
                    Ext.example.msg('Item Selected', 'You chose {0}.', rec.get('name'));
                }
            }
        }
});
```

将xtype指定为linechart，即在一个Panel中创建了折线图，图表的数据来自于上面定义好的store，xField指定横轴数据显示name列的数据，yField指定纵轴数据显示visits列的数据，最终的显示效果如图14-15所示。

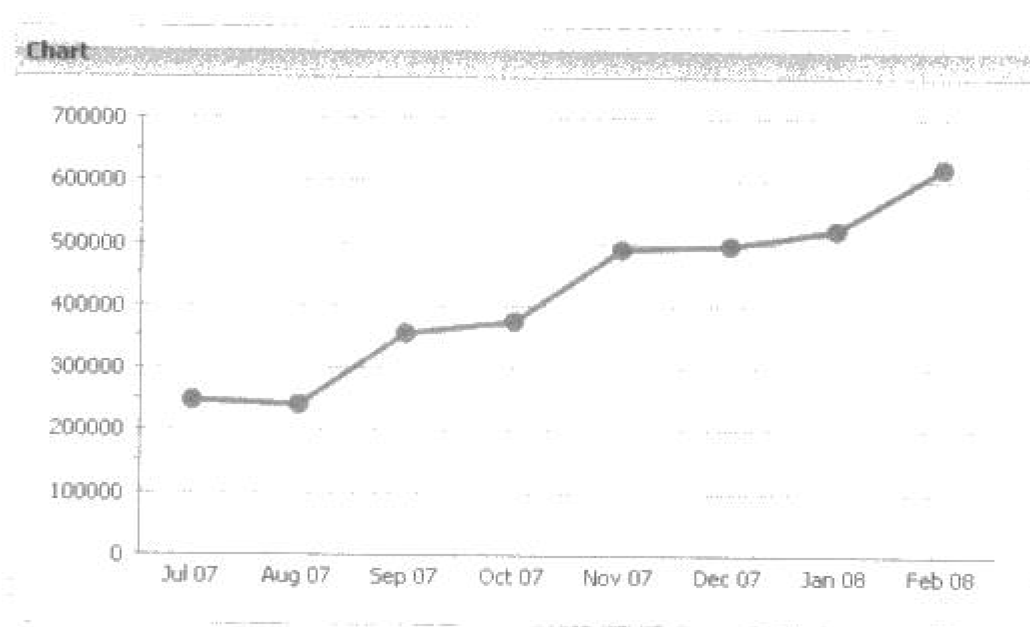


图14-15　折线图

14.4.4　饼状图

实现饼状图（PieChart）的代码如下所示：

```
new Ext.Panel({
    title: 'Chart',
    renderTo: 'container',
    width:500,
    height:300,
    layout:'fit',

    items: {
        xtype: 'piechart',
        url: '../../resources/charts.swf',
        store: store,
        categoryField: 'name',
```

```
            dataField: 'visits',
            listeners: {
                itemclick: function(o){
                    var rec = store.getAt(o.index);
                    Ext.example.msg('Item Selected', 'You chose {0}.', rec.get('name'));
                }
            }
        }
});
```

将xtype指定为barchart，即在一个Panel中创建了饼状图，图表的数据来自于上面定义好的store，categoryField指定类别数据显示name列的数据，dataField指定扇区数据显示vists列的数据，最终的显示效果如图14-16所示。

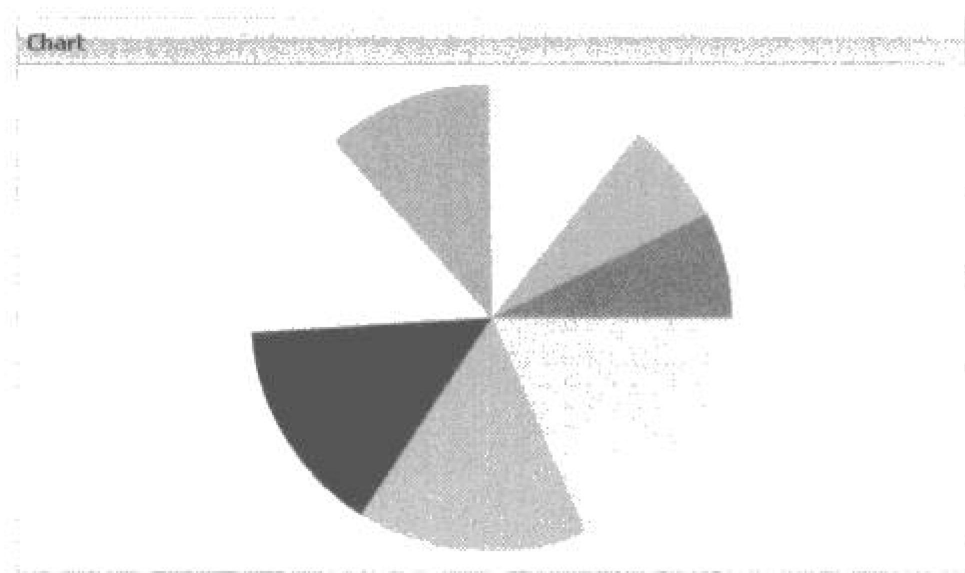


图14-16 饼状图

14.4.5 柱状栈图

实现柱状栈图（StackedColumnChart）的代码如下所示：

```
new Ext.Panel({
    title: 'Chart',
    renderTo: 'container',
    width:500,
    height:300,
    layout:'fit',

    items: {
        xtype: 'stackedcolumnchart',
        url: '../../resources/charts.swf',
        store: store,
        xField: 'name',
        series: [{
            yField: "visits",
            displayName: "visits:"
        }, {
```

```
                yField: "views",
                displayName: "views:"
            }],
            yAxis: new Ext.chart.NumericAxis({
                stackingEnabled: true,
                labelRenderer: Ext.util.Format.usMoney
            }),
            listeners: {
                itemclick: function(o){
                    var rec = store.getAt(o.index);
                    Ext.example.msg('Item Selected', 'You chose {0}.', rec.get('name'));
                }
            }
        }
    });
```

将xtype指定为stackedcolumnchart，即在一个Panel中创建了柱状栈图，图表的数据来自于上面定义好的store。xField指定横轴数据显示name列的数据，因为柱状栈图拥有多个纵轴数据，所以需要使用series代替原本柱状图中的yField来设置纵轴数据。这里设置了两部分数据visits和views，它们将以堆叠形式显示在纵轴部分，另外还要为yAxis添加stacking-Enabled:true参数，这样就完成了柱状栈图的设置。最终的显示效果如图14-17所示。

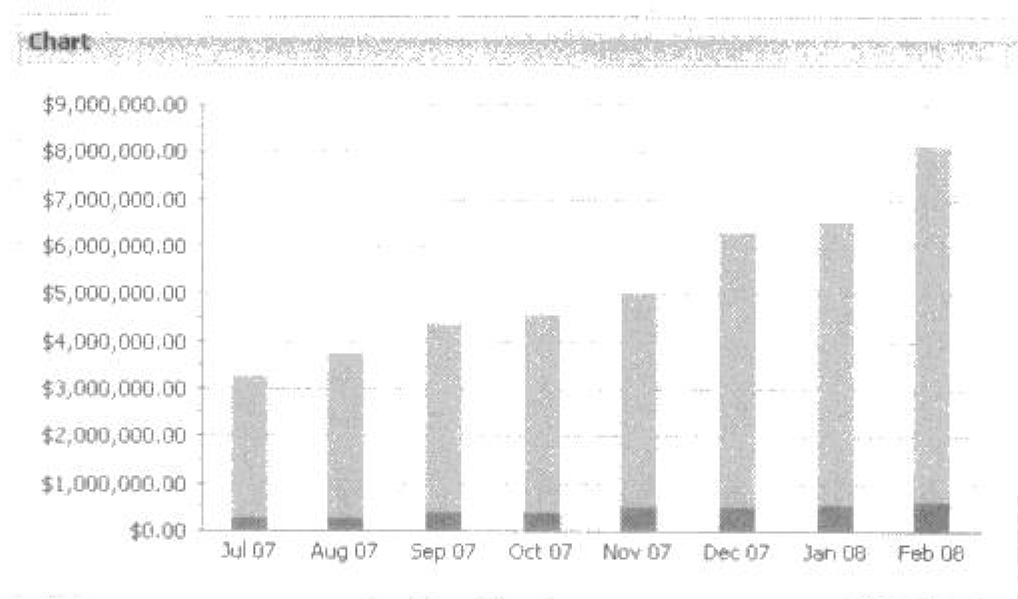


图14-17　柱状栈图

14.4.6　横向柱状栈图

实现横向柱状栈图（StackedBarChart）的代码如下所示：

```
new Ext.Panel({
    title: 'Chart',
    renderTo: 'container',
    width:500,
    height:300,
    layout:'fit',
```

```
        items: {
            xtype: 'stackedbarchart',
            url: '../../resources/charts.swf',
            store: store,
            yField: 'name',
            series: [{
                xField: "visits",
                displayName: "visits:"
            }, {
                xField: "views",
                displayName: "views:"
            }],
            xAxis: new Ext.chart.NumericAxis({
                stackingEnabled: true,
                labelRenderer: Ext.util.Format.usMoney
            })
        }
    });
```

将xtype指定为stackedbarchart，即在一个Panel中创建了横向柱状栈图，图表的数据来自于上面定义好的store。yField指定横轴数据显示name列的数据，因为横向柱状栈图拥有多个横轴数据，所以需要使用series代替原本横向柱状图中的xField来设置横轴数据。这里设置了两部分数据visits和views，它们将以堆叠形式显示在横轴部分，另外还要为xAxis添加stackingEnabled:true参数，这样就完成了横柱状栈图的设置。最终的显示效果如图14-18所示。

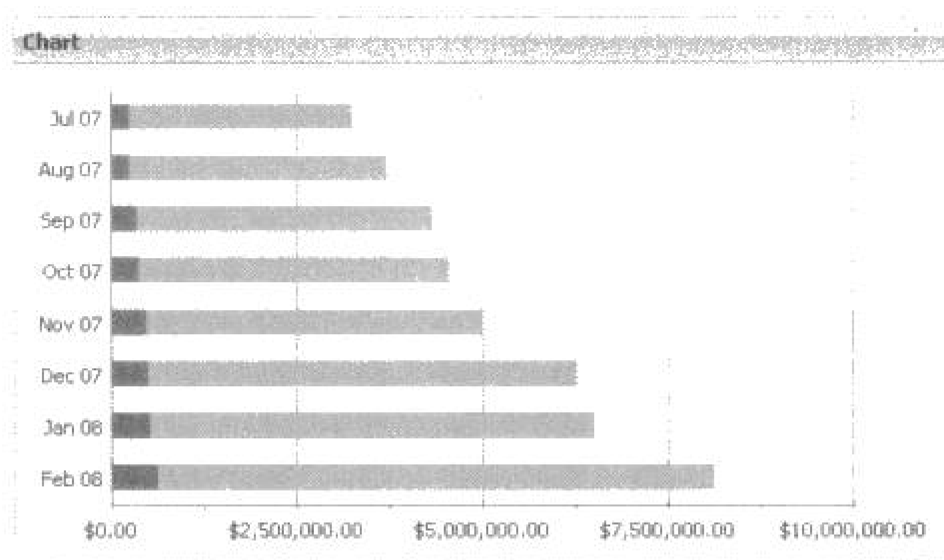


图14-18 横向柱状栈图

14.4.7 混合图

我们也可以通过设置不同类型的series实现混合图，实例代码如下所示：

```
new Ext.Panel({
    title: 'Chart',
```

```
        renderTo: 'container',
        width:500,
        height:300,
        layout:'fit',

        items: {
            xtype: 'stackedcolumnchart',
            url: '../../resources/charts.swf',
            store: store,
            xField: 'name',
            series: [{
                type: 'column',
                yField: "views",
                displayName: "views:"
            }, {
                type: 'line',
                yField: "visits",
                displayName: "visits:"
            }],
            yAxis: new Ext.chart.NumericAxis({
                stackingEnabled: true,
                labelRenderer: Ext.util.Format.usMoney
            })
        }
    });
```

上述代码为series中的不同部分设置了不同的类型，type:'column'表示views列的数据将以柱状图的形式显示，type:'line'表示visits列的数据将以折线图的显示，最终的显示效果如图14-19所示。

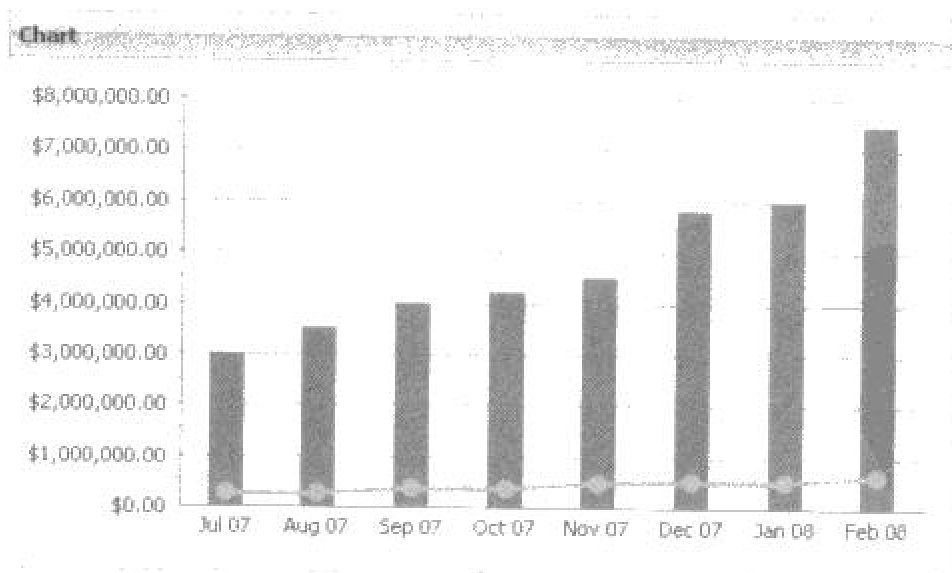


图14-19　混合图

14.5　EXT 3.1 带来的新特性

EXT 3.1的发布比预期推迟了一个月左右，但这段时间的等待是绝对值得的，因为这次版本

更新不但为整体框架实现了极高的性能提升，还为我们带来了大量奇妙的高级组件。下面将逐一进行介绍。

14.5.1 解决内存泄露

使用EXT之类的富客户端框架进行开发，一般都会采用one page one application的开发方式，一方面因为ext-all.js的体积有点儿大，页面刷新会感觉有点儿卡，另一方面因为大量使用Ajax不需要刷新页面就可以实现与后台交互。

但是，one page one application的开发方式也会造成一定问题，最明显的问题就是内存泄露。在同一个页面中重复进行多次DOM操作后，平时注意不到的内存损耗会慢慢积累起来，逐渐占据浏览器的内存，导致整体应用越来越慢。

EXT 3.1终于带来了官方提供的内存泄露解决方案，特别是这些解决方案可以在IE 6+系列的浏览器下发挥极大作用。

1. 解决DOM泄露

DOM泄露造成的问题一直是困扰开发者的大问题，每次在我们使用完一个组件后而没有调用`destroy()`函数进行销毁时，就会造成DOM泄露。EXT动态创建的DOM会一直驻留在内存中，无法被内存管理器回收。实际上在旧版本的EXT中，即使手工调用了`destroy()`函数，也有可能出现对应DOM没有被正确删除，从而造成DOM泄露的问题。

为此，EXT 3.1对组件进行了大量的重构，可以确保无论在ext-all或是在ext-core中都可以正确删除所有组件的DOM引用，避免DOM泄露问题。特别是在IE浏览器下，也可以确保将所有孤立的DOM节点都删除。

2. 解决JavaScript引擎造成的内存泄露

因为JavaScript引擎本身的缺陷导致的内存泄露，听起来让人有点儿吃惊，可现实是目前所有版本的IE浏览器（包括IE 8）都不能很好地处理JavaScript中长期驻留的对象和对象缓存，这就导致了大量的内存泄露。目前EXT 3.1中采用的解决办法是，当用户使用IE浏览器时主动刷新垃圾收集器，从而对无用的对象内存进行回收。通过测试，这样可以在长期运行的页面应用下获得50倍的效率提升。

图14-20展示了Ext JS官方提供的一份3.0版本与3.1版本长期打开一个页面应用的内存消耗情况。





图14-20 EXT 3.0与EXT 3.1内存消耗比较

14.5.2 核心组件优化

为了进一步提升框架的性能，EXT 3.1还对一些核心组件进行了优化，包括优化布局、重构事件管理器和提高Ext Core灵活性。

1. 优化布局

为了减少复杂布局中可能出现的大量计算，EXT 3.1中尽可能避免了上级容器重新布局时所产生的计算，比如EXT 3.1中直到整体布局完成之后才会计算子容器的大小。这样就可以大大减少布局中所需要的计算，用官方的说法就是："Layout Storm"（布局风暴）。

2. 重构事件管理器

EXT 3.1对事件管理器（EventManager）和对应的核心适配器（Adapter）进行了全面的代码重构，通过清除重复和执行效率比较低的代码来实现性能优化。之前的事件缓存机制也经过了全面的调整，可以为用户提供更快速和舒适的体验。

这次重构可以为处理复杂容器的场景提升一个数量级的响应速度。而在删除元素或组件时，EventManager又可以使用缓存和事件延迟的方式来避免外来的操作破坏正在运行的操作。

3. 提高Ext Core的灵活性

在EXT 3.0分拆Ext Core时，Ext Core的代码中包含了很多不容易被外部扩展的私有函数。为了顺应Ext社区对高灵活性的要求，EXT 3.1中开放了很多函数的原型。并且现在`Ext .each()`函数也更多地利用了底层库的代码，从而实现了很多常见场景下的性能提升。

14.5.3 分组表头

借助分组表头（Grouping GridView）这个组件，我们可以实现内容更加复杂的表格组件，如图14-21所示。

分组表头

Ext JS-1.x		Ext JS-2.x		Ext JS-3.x	
ext11	ext12	ext21	ext22	ext31	ext32
1	name 1	male	true	male	true
2	name 2	female	false	female	false
3	name 3	male	false	male	false
4	name 4	female	false	female	false
5	name 5	male	true	male	true

图14-21 分组表头

为了实现图14-21中分组表头的效果，我们首先要在页面中引入ColumnHeaderGroup.js和ColumnHeaderGroup.css，这两个文件就放在EXT 3.1发布包的examples/ux/目录下，代码如下所示：

```
<script type="text/javascript" src="../../ux/ColumnHeaderGroup.js"></script>
<link rel="stylesheet" type="text/css" href="../../ux/css/ColumnHeaderGroup.css" />
```

下一步是定义分组表头的配置。我们将定义一个两层的表头，每个上层表头下面包含两列，代码如下所示：

```
var row = [
```

```
        {header: 'Ext JS-1.x', colspan: 2, align: 'center'},
        {header: 'Ext JS-2.x', colspan: 2, align: 'center'},
        {header: 'Ext JS-3.x', colspan: 2, align: 'center'}
    ];

    var group = new Ext.ux.grid.ColumnHeaderGroup({
        rows: [row]
    });
```

接下来只要向GridPanel中加入上面配置好的组件就可以了，代码如下所示：

```
var grid = new Ext.grid.GridPanel({
    title: 'Column Header Group',
    width: 620,
    autoHeight: true,
    store: new Ext.data.SimpleStore({
        data: [
            [1, 'name 1', 'male',   'true',  'male',   'true'],
            [2, 'name 2', 'female', 'false', 'female', 'false'],
            [3, 'name 3', 'male',   'false', 'male',   'false'],
            [4, 'name 4', 'female', 'false', 'female', 'false'],
            [5, 'name 5', 'male',   'true',  'male',   'true']
        ],
        fields: ['ext11', 'ext12', 'ext21', 'ext22', 'ext31', 'ext32']
    }),
    columns: [
        {header: 'ext11', dataIndex: 'ext11'},
        {header: 'ext12', dataIndex: 'ext12'},
        {header: 'ext21', dataIndex: 'ext21'},
        {header: 'ext22', dataIndex: 'ext22'},
        {header: 'ext31', dataIndex: 'ext31'},
        {header: 'ext32', dataIndex: 'ext32'}
    ],
    renderTo: 'grid',
    plugins: group
});
```

注意，store中的字段设置以及表格中每一列的对应关系。这里我们一共配置了6列，ColumnHeaderGroup插件会根据我们上面进行的配置将对应的列合并在分组表头中。

实例代码参考14.core\ext31\1-groupGridView.html。

14.5.4　锁定列

锁定列（Locking Grid Column）的功能最初是在EXT 1.x中实现的，进入EXT 2.x时代后，因为列锁定功能实现过于复杂而被去掉了，但是社区中依然有人实现了锁定列的扩展。官方自然不甘落后，在EXT 3.1中提供了锁定列的功能。

在锁定一列之后，即使用户水平滚动表格，被锁定的表格也会不受影响地一直显示在表格左侧，如图14-22所示。

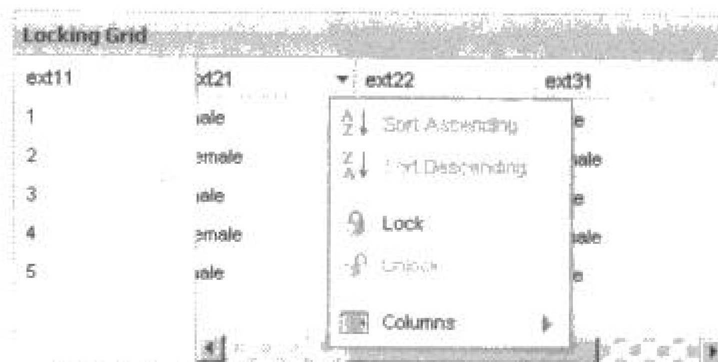


图14-22　锁定列

14

从图14-22中可以看到，我们可以在列头下拉菜单中控制对特定列的锁定和解锁，被锁定的列会自动显示在表格左侧，从而不会受到表格水平滚动条的影响。

为了实现这样一个拥有锁定某一列功能的表格，我们首先要在页面中引入对应的LockingGridView.js和LockingGridView.css，代码如下所示：

```
<script type="text/javascript" src="../../ux/LockingGridView.js"></script>
<link rel="stylesheet" type="text/css" href="../../ux/css/LockingGridView.css" />
```

这样就可以在代码中使用Ext.ux.grid.LockingColumnModel和Ext.ux.grid.LockingGridView两个类了。借助这两个类我们才能在表格中实现锁定列的功能，并且这两个列必须结合使用，否则会在运行时出现错误，导致整体表格无法正常使用。

为GridPanel添加锁定列功能的配置如以下代码所示：

```
var grid = new Ext.grid.GridPanel({
    title: 'Locking Grid',
    width: 400,
    height: 200,
    store: new Ext.data.SimpleStore({
        data: [
            [1, 'name 1', 'male',   'true',  'male',   'true'],
            [2, 'name 2', 'female', 'false', 'female', 'false'],
            [3, 'name 3', 'male',   'false', 'male',   'false'],
            [4, 'name 4', 'female', 'false', 'female', 'false'],
            [5, 'name 5', 'male',   'true',  'male',   'true']
        ],
        fields: ['ext11', 'ext12', 'ext21', 'ext22', 'ext31', 'ext32']
    }),
    colModel: new Ext.ux.grid.LockingColumnModel([
        {header: 'ext11', dataIndex: 'ext11'},
        {header: 'ext12', dataIndex: 'ext12'},
        {header: 'ext21', dataIndex: 'ext21'},
        {header: 'ext22', dataIndex: 'ext22'},
        {header: 'ext31', dataIndex: 'ext31'},
        {header: 'ext32', dataIndex: 'ext32'}
    ]),
    renderTo: 'grid',
    view: new Ext.ux.grid.LockingGridView()
});
```

如果希望将某些列在表格最初显示时就锁定上，可以直接在colModel的配置中添加locked: true属性，这一列就会在初始化时锁定在表格左侧。

实例代码参考14.core\ext31\2-lockingGrid.html。

14.5.5 树形表格

树形表格（TreeGrid）同时具备树形的分级结构和表格的丰富内容。第5章已经介绍了以树形为基础实现的Ext.ux.tree.ColumnTree扩展组件，现在EXT 3.1又我们带来了基于表格的树形表格，如图14-23所示。

树形表格

Task	Duration	Assigned To
Project: Shopping	13.25	Tommy Maintz
Housewares	1.25	Tommy Maintz
Remodeling	12	Tommy Maintz
Paint bedroom	2.75	Tommy Maintz
Decorate living room	2.75	Tommy Maintz
Fix lights	0.75	Tommy Maintz
Reattach screen door	2	Tommy Maintz
Retile kitchen	6.5	Tommy Maintz
Project: Testing	2	Core Team

图14-23 树形表格

实际上树形表格也是借助Ext.tree.TreeLoader实现加载树形数据的。使用树形表格之前，要先在页面中引入对应的JavaScript脚本和CSS样式表，代码如下所示：

```
<script type="text/javascript" src="../../ux/treegrid/TreeGridSorter.js"></script>
<script type="text/javascript" src="../../ux/treegrid/TreeGridColumnResizer.js"></script>
<script type="text/javascript" src="../../ux/treegrid/TreeGridNodeUI.js"></script>
<script type="text/javascript" src="../../ux/treegrid/TreeGridLoader.js"></script>
<script type="text/javascript" src="../../ux/treegrid/TreeGridColumns.js"></script>
<script type="text/javascript" src="../../ux/treegrid/TreeGrid.js"></script>
<link rel="stylesheet" type="text/css" href="../../ux/treegrid/treegrid.css" rel=
"stylesheet" />
```

下面直接创建树形表格就可以了，详细配置代码如下所示：

```
var tree = new Ext.ux.tree.TreeGrid({
    title: 'Tree Grid',
    width: 500,
    height: 300,
    renderTo: 'grid',
    enableDD: true,

    columns:[{
        header: 'Task',
        dataIndex: 'task',
        width: 230
    },{
        header: 'Duration',
        width: 100,
        dataIndex: 'duration',
        align: 'center'
    },{
        header: 'Assigned To',
        width: 150,
        dataIndex: 'user'
    }],

    dataUrl: 'treegrid-data.json'
});
```

页面初始化后，树形表格会去treegrid-data.json获取显示所需的数据，我们将其中一段内容截取出来放在下面：

```
[{
    task:'Project: Shopping',
    duration:13.25,
    user:'Tommy Maintz',
    iconCls:'task-folder',
    expanded: true,
    children:[{
        task:'Housewares',
        duration:1.25,
        user:'Tommy Maintz',
        iconCls:'task-folder',
        children:[{
            task:'Kitchen supplies',
            duration:0.25,
            user:'Tommy Maintz',
            leaf:true,
            iconCls:'task'
        }
```

从上述代码中可以看到，task、duration、user的数据会直接显示在表格中，而children和expaned两个属性实现了树形结构所需的上下层关系，以及下一级节点是否默认展开的配置。我们可以把这些数据看做在每个节点中添加了额外信息的树形数据。

实例代码参考14.core\ext31\3-treeGrid.html。

如果还想比较一下树形表格与Ext.ux.tree.ColumnTree，可以参考5.13节。

14.5.6　竖直布局

竖直布局（VBox）与水平布局（HBox）的用法十分类似，可以为一列组件提供统一的布局控制，如图14-24所示。

与上面介绍的组件不同，竖直布局已经作为EXT 3.1默认发布包的一部分了，使用之前不需要额外引入其他脚本和样式文件。

为实现上面竖直布局的效果，可以使用如下配置代码：

```
var panel = new Ext.Panel({
    title: 'VBox',
    width: 200,
    height: 400,
    renderTo: 'grid',
    layout: {
        type:'vbox',
        padding:'5',
        align:'stretch'
    },
    defaults:{margins:'0 0 5 0'},
    items:[{
        xtype:'button',
        text: 'Button 1',
```

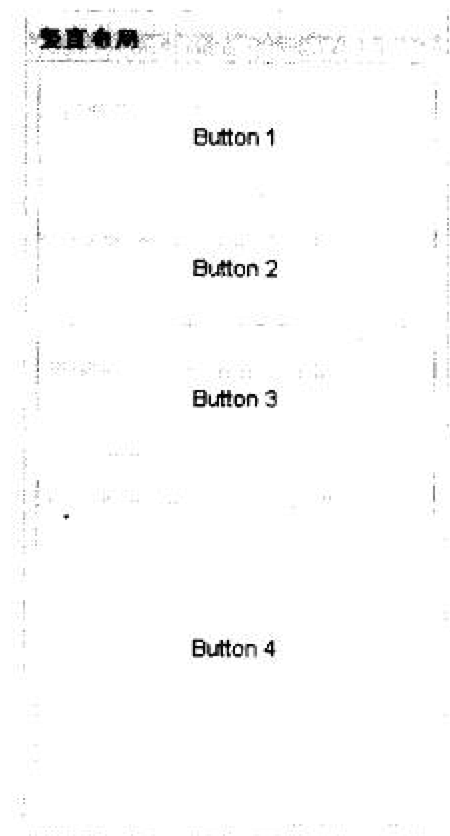


图14-24　竖直布局

```
            flex:1
        },{
            xtype:'button',
            text: 'Button 2',
            flex:1
        },{
            xtype:'button',
            text: 'Button 3',
            flex:1
        },{
            xtype:'button',
            text: 'Button 4',
            flex:3,
            margins:'0'
        }]
    });
```

与布局相关的配置放在layout属性中，我们用type:'vbox'指定当前的Panel使用竖直布局方式。可以为其中每个组件设置flex属性，flex属性越大，对应的组件就会占据越大空间。

垂直布局还支持使用align属性对布局中的组件设置统一的对齐方式，比如上例中将align属性设置为'stretch'，就会将Panel内部的组件宽度都自动设置为充满外部容器的大小，还可以将align属性设置为'left'、'center'、'stretchmax'等值。

实例代码参考14.core\ext31\4-vbox.html。

竖直布局方式与EXT 3.x之前的水平布局方式十分相似，如果想了解水平布局方式可以参考8.11节。

14.5.7 高级表格查询

高级表格查询（GridFiltering）早在EXT 2.x时代就作为第三方扩展组件出现在社区中了，如今官方继续将此功能加强，并随EXT 3.1版本一起发布。

高级表格查询显示效果如图14-25所示。

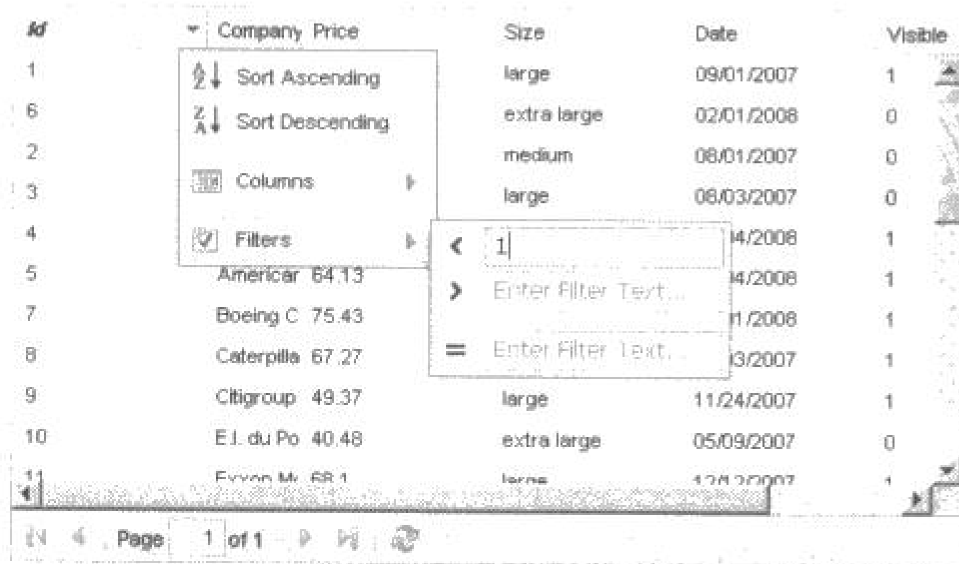


图14-25 高级表格查询

14

使用高级表格查询功能，首先要在页面中引入所有所需的扩展组件，如下面代码所示：

```
<script type="text/javascript" src="../../ux/gridfilters/menu/RangeMenu.js"></script>
<script type="text/javascript" src="../../ux/gridfilters/menu/ListMenu.js"></script>

<script type="text/javascript" src="../../ux/gridfilters/GridFilters.js"></script>
<script type="text/javascript" src="../../ux/gridfilters/filter/Filter.js"></script>
<script type="text/javascript" src="../../ux/gridfilters/filter/StringFilter.js"></script>
<script type="text/javascript" src="../../ux/gridfilters/filter/DateFilter.js"></script>
<script type="text/javascript" src="../../ux/gridfilters/filter/ListFilter.js"></script>
<script type="text/javascript" src="../../ux/gridfilters/filter/NumericFilter.js"></script>
<script type="text/javascript" src="../../ux/gridfilters/filter/BooleanFilter.js"></script>
<link rel="stylesheet" type="text/css" href="../../ux/gridfilters/css/GridFilters.css" />
<link rel="stylesheet" type="text/css" href="../../ux/gridfilters/css/RangeMenu.css" />
```

下面可以为表格构建高级查询组件了，如下面代码所示：

```
var filters = new Ext.ux.grid.GridFilters({
    filters: [{
        type: 'numeric',
        dataIndex: 'id'
    }, {
        type: 'string',
        dataIndex: 'company',
        disabled: true
    }, {
        type: 'numeric',
        dataIndex: 'price'
    }, {
        type: 'date',
        dataIndex: 'date'
    }, {
        type: 'list',
        dataIndex: 'size',
        options: ['small', 'medium', 'large', 'extra large'],
        phpMode: true
    }, {
        type: 'boolean',
        dataIndex: 'visible'
    }]
});
```

下面将filters设置到表格中，就完成了高级表格查询功能的配置，代码如下所示：

```
var grid = new Ext.grid.GridPanel({
    width: 500,
    height: 300,
    border: true,
    store: store,
    renderTo: 'grid',
    colModel: new Ext.grid.ColumnModel({
        columns: [{
            dataIndex: 'id',
            header: 'Id',
            filterable: true
```

```
        }, {
            dataIndex: 'company',
            header: 'Company',
            id: 'company',
            filter: {
                type: 'string'
            }
        }, {
            dataIndex: 'price',
            header: 'Price',
            filter: {
            }
        }, {
            dataIndex: 'size',
            header: 'Size',
            filter: {
                type: 'list',
                options: ['small', 'medium', 'large', 'extra large']
            }
        }, {
            dataIndex: 'date',
            header: 'Date',
            renderer: Ext.util.Format.dateRenderer('m/d/Y'),
            filter: {
            }
        }, {
            dataIndex: 'visible',
            header: 'Visible',
            filter: {
            }
        }],
        defaults: {
            sortable: true
        }
    }),
    loadMask: true,
    plugins: [filters],
    autoExpandColumn: 'company',
    bbar: new Ext.PagingToolbar({
        store: store,
        pageSize: 50,
        plugins: [filters]
    })
});
```

注意要将`filters`同时设置为表格和PagingToolbar的插件（plugin），这样才能在进行分页时也不会丢失原来设置的查询信息。

实例代码参考14.core\ext31\5-gridfiter.html。

14.5.8 自定义编辑器

EXT中提供的自定义编辑器（Editor）可以实现自由编辑任何一个HTML标签中的内容，我

们完全可以在页面中随便写几个标签，然后为这些标签统一注册一个自定义编辑器进行编辑。下面我们就在页面中设置两个div标签，并在用户使用鼠标双击标签时显示自定义编辑器对标签内容进行编辑。效果如图14-26所示。



图14-26 自定义编辑器

首先要在页面上添加两个div标签，代码如下所示：

```
<div style="margin:50px;">
    <div style="width:200px;float:left;">text1</div>
    <div style="width:200px;float:left;">text2</div>
</div>
```

然后在JavaScript代码中设置监听body部分的鼠标双击事件，并在内层嵌套的div标签中触发鼠标双击事件时对标签内容进行编辑，代码如下所示：

```
var editor = new Ext.Editor({
    shadow: false,
    completeOnEnter: true,
    cancelOnEsc: true,
    updateEl: true,
    ignoreNoChange: true,
    alignment: 'l-l',
    field: {
        allowBlank: false,
        xtype: 'textfield',
        width: 90,
        selectOnFocus: true
    }
});

Ext.getBody().on('dblclick', function(e, t){
    editor.startEdit(t);
}, null, {
    delegate: 'div div'
});
```

实例代码参考14.core\ext31\6-editor.html。

14.6 EXT 3.2 带来的新特性

EXT官方开发团队于2010年3月30日发布了EXT 3.2，我们可以在官方网站上免费下载发布包。据官方消息，这次发布修复了180多个bug，并在整体框架级别都实现了很大的提升。

自从EXT 1.0发布以来，EXT每次发布的新版本都会为我们带来绚丽的界面效果。正当EXT一步步走向成熟之时，我们不禁会问，EXT所创造的界面奇迹是否已经登峰造极，难以超越了呢？EXT 3.2的发布彻底打消了这种疑虑，它用更加难以置信的界面效果再次证明了自己的完美。下面就让我们一起来领略一下EXT 3.2带来的新功能。

14.6.1 多重排序

在之前的版本中，我们只能对store中的某一列进行排序。EXT 3.2冲破了这种限制，现在我们可以为store中的多列同时指定排序规则了，如图14-27所示。

图14-27演示了同时对表格中的Rating与Salary两列进行排序，其中Rating列采用倒序排列，Salary列采用正序排列。多重排序功能主要通过store的sort()函数实现，具体用法如下列代码所示：

多重排序

Sorting order: Rating Salary

Name	Rating	Salary
name1	1	$10,000.10
name2	2	$11,111.10
name3	2	$2,222.20
name4	3	$44,444.40
name5	3	$55.30

图14-27 多重排序

```
store.sort([{
    field: 'rating',
    direction: 'DESC'
}, {
    field: 'salary',
    direction: 'DESC'
}]);
```

上面的代码，在排序时同时指定了两列的排序方式，store会在排序时优先使用前面的条件进行排序，当出现数值相同的情况时，再使用后面的条件对重复的数据进行处理。

14.6.2 为 DataView 添加动画变换效果

DataView可以实现多种页面效果的描绘，EXT 3.2带来的Ext.ux.DataViewTransition可以为DataView提供动画变换效果。当我们对store中的数据进行过滤时，Ext.ux.DataView-Transition就会将数据过滤的过程以动画形式显现出来，这会给用户带来极大的视觉冲击。实例效果如图14-28所示。



图14-28 为DataView添加动画变换效果

使用Ext.ux.DataViewTransition非常简单，只要将创建的实例设置为DataView的一个插件就可以了，如下面代码所示：

```
plugins : [
    new Ext.ux.DataViewTransition({
        duration  : 550,
        idProperty: 'id'
    })
],
```

14.6.3 组合表单控件

EXT 3.2中新增的组合表单控件是此次版本发布的另一大亮点，它解决了长久以来EXT难以实现复杂表单布局的问题，现在我们可以很轻松地在表单中实现多种多样的布局组合了，实例效果如图14-29所示。



图14-29　组合表单控件

图14-29中实现了一个标签对应三个文本框的效果，而且这三个文本框还实现了横排。这在以前都要通过手工控制分列布局，而且要编写大量的代码。如果采用EXT 3.2的组合表单控件，只需要编写如下代码，就可以实现这种效果。

```
new Ext.form.FormPanel({
    title: '组合表单控件',
    renderTo: 'docbody',
    width: 300,
    items: [{
        xtype: 'compositefield',
        fieldLabel: 'Full Name',
        items: [
            {xtype: 'textfield', name: 'title',     width: 40},
            {xtype: 'textfield', name: 'firstName', flex : 1},
            {xtype: 'textfield', name: 'lastName',  flex : 2}
        ]
    }],
    buttons: [{
        text: 'submit'
    }, {
        text: 'reset'
    }]
});
```

在上面代码中，我们直接通过xtype指定了compositefield，可以像使用其他普通输入控件一样为它指定标签和其他属性，组合输入控件中的子控件都包含在items中，默认会使用水平布局处理这些子控件的排列，既可以为它们指定宽度，也可以使用flex让它们自动适应容器的大小。

14.6.4 滑动条表单控件

顾名思义，滑动条表单控件就是可以将滑动条作为一个表单组件放在表单面板中进行布局，实现表单数据的修改、读取与提交功能，实例效果如图14-30所示。

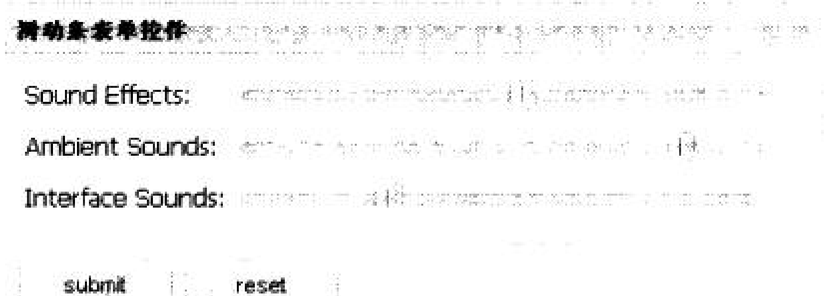


图14-30　滑动条表单控件

为使用滑动条表单控件，首先要在页面中引入SlideField.js扩展，然后就可以在表单面板中使用了，使用的方式与其他普通表单控件一致，实例代码如下所示：

```
var form = new Ext.form.FormPanel({
    width : 400,
    height: 160,
    title : '滑动条表单控件',

    bodyStyle   : 'padding: 10px;',
    renderTo    : 'container',
    defaultType: 'sliderfield',
    buttonAlign: 'left',

    defaults: {
        anchor: '95%',
        tipText: function(thumb){
            return String(thumb.value) + '%';
        }
    },
    items: [{
        fieldLabel: 'Sound Effects',
        value: 50,
        name: 'fx'
    },{
        fieldLabel: 'Ambient Sounds',
        value: 80,
        name: 'ambient'
    },{
        fieldLabel: 'Interface Sounds',
        value: 25,
        name: 'iface'
    }],
    buttons: [{
        text: 'submit'
    }, {
        text: 'reset'
    }]
});
```

14.6.5　为滑动条指定多个滑块

在EXT 3.2中，我们可以在同一个滑动条上指定多个滑块，让用户可以同时指定多个数值。显示多个滑块的显示效果如图14-31所示。

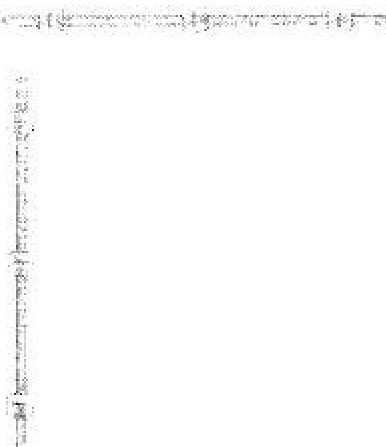


图14-31　为滑动条指定多个滑块

图14-31中效果的实现方式十分简单，只需要在创建滑动条的时候为它的value属性指定一个数组就可以了，实现代码如下所示：

```
var horizontal = new Ext.Slider({
    renderTo: 'multi-slider-horizontal',
    width   : 214,
    minValue: 0,
    maxValue: 100,
    values  : [10, 50, 90],
    plugins : new Ext.slider.Tip()
});

var vertical = new Ext.Slider({
    renderTo : 'multi-slider-vertical',
    vertical : true,
    height   : 214,
    minValue: 0,
    maxValue: 100,
    values  : [10, 50, 90],
    plugins : new Ext.slider.Tip()
});

Ext.get('btn').on('click', function() {
    Ext.Msg.alert('info', horizontal.getValues() + '|' + vertical.getValues());
});
```

在上面代码中，我们创建了两个滑动条，它们分别用水平和竖直两种渲染方式。我们使用数组为每个滑动条指定了三个数值，这样在滑动条上面就会在对应位置上显示三个滑块。在用户拖动滑块进行修改之后，还可以使用getValues()函数获得修改后的数值，返回的结果将是一个字符串，其中每个滑块对应的数值会以逗号进行分隔。如果对上例中的一个活动条调用getValues()函数，那么返回的结果将是"10,50,90"。

14.6.6　更多工具条插件

EXT 3.2提供了更多与工具条相关的插件，可以大大增强工具条的实用性与易用性。比如，排序插件（reorderer）支持使用拖曳方式对工具条上的元素进行排序，拖曳插件（dropable）可以

将页面上的其他内容拖曳到工具条上，从而创建新的工具条元素。

排序插件的效果如图14-32所示。



图14-32 工具条排序插件

如果希望使用排序插件，首先要在页面中引入Reorderer.js和ToolbarReorderer.js两个文件，之后创建一个`ToolbarReorderer`实例，将此实例设置为工具条的插件就可以了，实现代码如下所示：

```
plugins : [
    new Ext.ux.ToolbarReorderer({
        defaultReorderable: true
    })
],
```

拖曳插件的效果如图14-33所示。

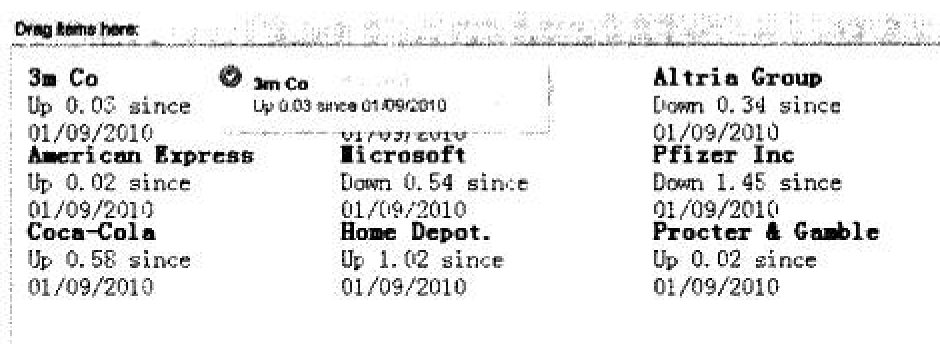


图14-33 工具条拖曳插件

如果想使用拖曳插件，首先要在页面中引入ToolbarDroppable.js，之后创建一个`ToolbarDroppable`实例，并实现其中的`createItem()`函数，这个函数就是拖曳成功后用于创建工具条元素的回调函数，最后将此实例设置为工具条的插件就可以了，实现代码如下所示：

```
var toolbar = new Ext.Toolbar({
    renderTo: 'docbody',
    plugins : [
        new Ext.ux.ToolbarDroppable({
            createItem: function(data) {
                var record = data.draggedRecord;

                return new Ext.Button({
                    text    : record.get('company'),
                    iconCls: record.get('change') > 0 ? 'money-up' : 'money-down',
                    reorderable: true
                });
            }
        })
    ],
    items: ['Drag items here:']
});
```

实例中使用了DataView和DragZone来实现从页面拖曳内容到工具条上的效果，可以参考光盘中的实例代码。

14.6.7　新主题 Accessibility

EXT 3.2中提供了用于支持《美国残疾人康复法案》第508节的新主题——Accessibility，这个新主题在深色背景上提供了更大的文字和强烈对比的颜色，这是为了帮助视障用户而使用的高对比度方案。

如果想使用这个新主题，可以直接在页面中引入xtheme-access.css，新主题效果如图14-34所示。

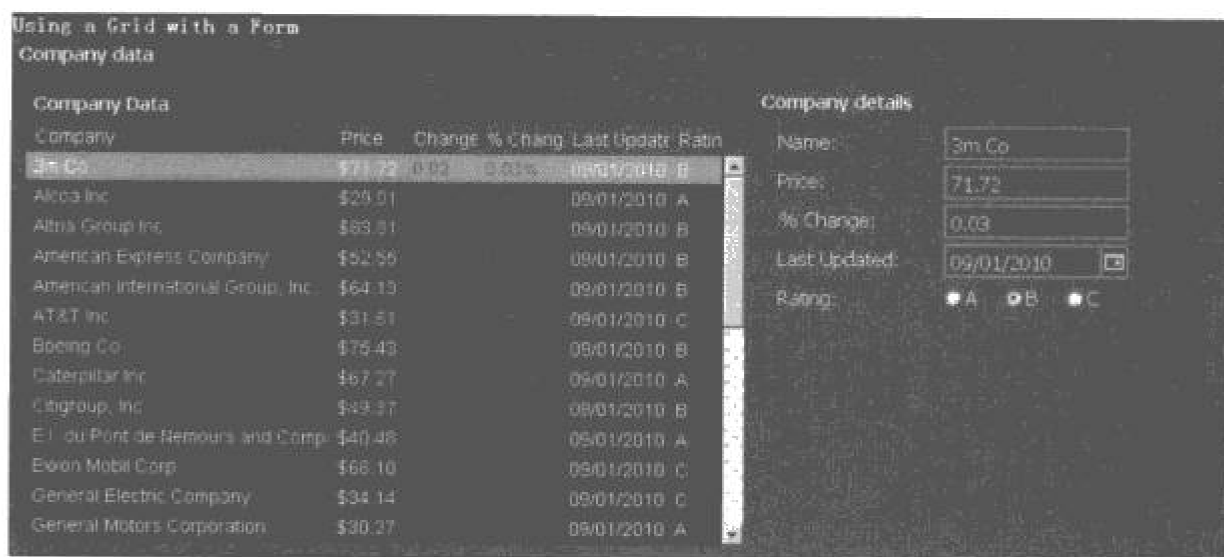


图14-34　新主题Accessibility

14.7　小结

本章主要介绍了随EXT 3.0的正式发布而来的诸多功能与组件。

Ext Core抽取了EXT中最核心的功能与组件，其中剔除了原来EXT中复杂的数据操作和众多组件，我们完全可以在Ext Core的基础上直接建立完整的网络应用。

Ext Direct提出了前台JavaScript与后台服务交互数据的标准，借助不同平台之上的支持库，我们可以让同一套EXT编写的前台应用无需任何修改便可以运行在所有平台上。

EXT 3.0中提供的各种组件足以让任何人眼前一亮。更令人惊讶的是，RowEditor或者任何一个分页小组件都可以直接以插件的形式组装到现有系统中，无需改动其他代码。

Flash图表组件是另一个重大突破，JavaScript的图形方面一直受限于浏览器的支持，Flash图表的加入可以让我们在页面上将数据以华丽图表的形式呈现出来。

还介绍了EXT 3.1版本中对整体框架性能方面实现的巨大提升，并介绍了其新添加的新组件，包括分组表头、锁定列、树形表格、竖直布局、高级表格查询和自定义编辑器。

本章最后讲述了最新发布的EXT 3.2中展示的最新组件，包括多重排序功能、DataView中的动画效果、组合表单控件、滑动条表单控件、在活动条中使用多个滑块、两个工具条插件和新主题Accessibility。

第 15 章

用户扩展与插件

15

本章内容

- 介绍用户扩展
- 编写用户扩展所需的基础知识
- 编写自定义用户扩展
- 介绍EXT的插件体系

EXT本身拥有极强的扩展性，它允许用户在其基础上编写更适合自身业务需要的自定义组件，因此学习如何为EXT编写扩展组件是一件非常有意义的事情。我们可以将实际开发中常用的功能封装为自定义组件，下次再遇到类似的功能时就可以直接拿来使用，不必重复编写代码。

15.1 介绍用户扩展

在EXT中，用户扩展组件被称为User Extension，默认放置在Ext.ux命名空间下。EXT的官方论坛上特地为用户扩展设立了专门的版块，在其中可以找到很多第三方扩展用户组件。实际上，EXT中所有的组件都可以看作官方提供的用户扩展。每次我们利用EXT中的继承机制从父类中获得一切默认功能，并添加上定制的业务功能时，就已经是在实现一个特定的用户扩展组件了。

下面我们利用官方提供的SearchField来带领大家认识一下用户扩展组件User Extension，SearchField.js放在发布包中的examples/ux目录下。

我们需要在页面中引入SearchField.js，实例代码如下所示：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>ux</title>
    <link rel="stylesheet" type="text/css" href="../../resources/css/ext-all.css" />
    <script type="text/javascript" src="../../adapter/ext/ext-base.js"></script>
    <script type="text/javascript" src="../../ext-all.js"></script>
    <link rel="stylesheet" type="text/css" href="../shared/examples.css" />

    <script type="text/javascript" src="../ux/SearchField.js"></script>
    <script type="text/javascript">
Ext.onReady(function() {
```

```
        var data = [
            ['name1', 'value1'],
            ['name1', 'value1']
        ];

        var store = new Ext.data.ArrayStore({
            proxy: new Ext.data.MemoryProxy(data),
            fields: ['name', 'value']
        });

        var field = new Ext.ux.form.SearchField({
            store: store,
            renderTo: 'container'
        });
    });
        </script>
      </head>
      <body>
        <script type="text/javascript" src="../shared/examples.js"></script>
        <div id="container"></div>
      </body>
    </html>
```

在使用Ext.ux.form.SearchField时需要绑定一个store，这样当用户输入搜索条件时就会调用对应store的load()函数，传入用户输入的查询条件进行搜索，如图15-1所示。

用户可以在输入框中输入查询条件，然后点击右侧的查询按钮，或者直接点击回车就可以执行搜索，如图15-2所示。



图15-1　SearchField的初始显示效果

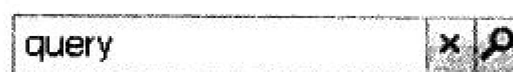


图15-2　输入查询条件之后的显示效果

SearchField在执行搜索之后就会显示出另外一个按钮，点击这个按钮就可以清空输入框中已存在的数据。在输入框中不存在数据时，这个清空按钮又会被隐藏，这对于用户体验是一个相当不错的例子。

现在我们可以来看一下SearchField.js中的代码，了解一下这个用户扩展组件是如何实现的，代码如下所示：

```
Ext.ux.form.SearchField = Ext.extend(Ext.form.TwinTriggerField, {
    initComponent : function(){
        Ext.ux.form.SearchField.superclass.initComponent.call(this);
        this.on('specialkey', function(f, e){
            if(e.getKey() == e.ENTER){
                this.onTrigger2Click();
            }
        }, this);
    },

    validationEvent:false,
```

```
        validateOnBlur:false,
        trigger1Class:'x-form-clear-trigger',
        trigger2Class:'x-form-search-trigger',
        hideTrigger1:true,
        width:180,
        hasSearch : false,
        paramName : 'query',

        onTrigger1Click : function(){
            if(this.hasSearch){
                this.el.dom.value = '';
                var o = {start: 0};
                this.store.baseParams = this.store.baseParams || {};
                this.store.baseParams[this.paramName] = '';
                this.store.reload({params:o});
                this.triggers[0].hide();
                this.hasSearch = false;
            }
        },

        onTrigger2Click : function(){
            var v = this.getRawValue();
            if(v.length < 1){
                this.onTrigger1Click();
                return;
            }
            var o = {start: 0};
            this.store.baseParams = this.store.baseParams || {};
            this.store.baseParams[this.paramName] = v;
            this.store.reload({params:o});
            this.hasSearch = true;
            this.triggers[0].show();
        }
    });
```

从上述代码中可以看到，Ext.ux.form.SearchField继承自Ext.form.TwinTriggerField，这个父类支持在数据框右侧设定两个功能按钮，然后分别使用onTrigger1Click()和onTrigger2Click()两个函数监听两个功能按钮的点击操作。

Ext.form.TwinTriggerField中的onTrigger1Click()和onTrigger2Click()都没有具体实现，需要在实际的子类中，比如我们现在看到的Ext.ux.form.SearchField，对两个回调函数进行实现。下面我们来看一下SearchField是如何通过扩展实现具体功能的。

首先在SearchField中覆盖了父类的initComponent()函数，在其中添加了对回车键的支持。在用户输入完查询信息后按下回车时，就会调用SearchField中的onTrigger2Click ()函数。

下面可以看到SearchField对父类中的一些参数进行了重设，比较重要的参数是hideTrigger1: true，这样组件初始化时就会隐藏第一个功能按键，也就形成了图15-1中的显示效果。在用户点击查询按钮或敲击回车键时，就会执行onTrigger2Click ()中的功能。这一步骤会判断当前的查询信息是否为空，如果为空就会调用onTrigger1Click()执行清空操作，

15

也就是说SearchField会自动判断是否拥有查询条件，选择执行查询或是清空操作。

在执行查询功能时，就会将用户输入的信息作为参数传递给已绑定的store，并调用对应的load()函数，然后还要将清空按钮显示出来。

在执行清空操作时，首先会清空输入框中的数据，然后隐藏SearchField的清空按钮，一切就是这么简单。

至此为止，我们已经实际看到了一个用户扩展是如何编写的，并将整个用户扩展实际应用起来。从这个例子中可以看到，当我们需要实现一个搜索输入框时并不需要实现所有功能操作，只需要通过继承Ext.form.TwinTriggerField获得基本功能，并覆盖需要扩展的部分功能，提供自定义实现。这样无疑可以大大降低开发工作量，同时又实现了很好的封装，保证在之后使用类似功能时可以直接调用。

实际上，在我们使用SearchField的过程中也可以了解到，所谓的用户扩展组件User Extension与其他官方提供的组件并没有任何不同，创建方式、参数的配置方式，以及方法的调用方式都是完全相同的。对开发者和应用者来说，所以实现用户扩展组件只会给我们带来极大的便利。

15.2 编写用户扩展所需的基础知识

在我们实际动手编写自己定制的用户扩展之前，首先要了解一些预备知识。实际上，也只有了解这些知识之后才能开始了解这些用户扩展组件时如何运作的，同时对EXT中的组件结构有了更如深入的了解，有助于我们对EXT进行更深一步的理解和研究。

15.2.1 继承模型

EXT中的用户扩展组件，是建立在面向对象开发的基础上的，所有的用户扩展都是基于EXT官方已经封装好的父类进行的功能扩展，因此我们首先需要了解的就是如何在EXT中实现面向对象编程。

EXT中利用Ext.extend()函数来实现类之间的继承操作。

在研究这个基础函数之前，我们还需要复习一下JavaScript的基本知识：prototype、constructor和closure。

prototype是在JavaScript中实现继承的基础，如果我们想在JavaScript中实现继承关系，可以使用如下代码：

```
RectAngle = function(width,height) {
    this.width = width;
    this.height = height;
}

RectAngle.prototype.area = function() {
    return this.width * this.height;
}
```

这段代码来自《JavaScript权威指南》，实现的功能其实很简单，它定义了一个“矩形”的构造函数，有长和宽两个参数。然后在RectAngle的prototype属性中添加一个计算面积的函数

area()。这样每次在执行var rect = new RectAngle();创建新的矩形对象后，都可以对rect对象调用area()参数了，因为rect对象从RectAngle的prototype里面继承了area()函数。

这就是JavaScript基于原型继承的简单理解。

constructor是JavaScript中定义的对象的构造函数。因为每个函数都会有一个prototype属性，构造函数也是一个函数，因此构造函数也拥有prototype属性。

prototype属性在定义函数时就会自动创建并初始化，也就是说，在执行RectAngle = function(width, height){}时，RectAngle中的prototype属性就已经被创建了，这时prototype中只包含一个树形，它就是constructor，这个constructor指向RectAngle函数本身。这样就形成了一个闭环，可以试验一下这种调用方式：

```
RectAngle.prototype.constructor.prototype.constructor…
```

通过上面代码这种调用方式，告诉我们可以通过不同的途径来调用RectAngle函数本身以及函数中的prototype属性。

对每个RectAngle创建的对象实例来说，例如var rect = new RectAngle(10, 10);，rect.prototype属性都会指向构造函数的prototype，也就是会指向RectAngle的prototype属性，因此RectAngle的所有实例对象都会共享同一份RectAngle.prototype属性。

这样一来就不需要分配多余的内存给每个对象实例来存储prototype属性，从而实现了JavaScript的继承功能。

closure含义是闭包，这是一种动态语言大多支持的简易功能，使用闭包我们可以更容易地实现各种匿名内部类。

```
RectAngle = function(width, height) {
    this.getWidth = function() {
        return width;
    };
    this.getHeight = function() {
        return height
    };
}

RectAngle.prototype.area = function() {
    return this.getWidth() * this.getHeight();
}
```

这段代码在构造函数中使用闭包为RectAngle对象定义了getWidth()和getHeight()两个函数。这两个函数都是在构造对象实例时创建的，因此不会通过prototype实现共享，也就是说每个对象实例都会分配不同的内存存储这些函数，也正因为如此不会出现各个实例之间的数据冲突，实际使用时可以根据现实需要进行选择。

下面我们可以来分析Ext.extend()的具体实现了，首先还是先来查看一下通常情况下如何实现JavaScript继承，代码如下所示：

```
RectAngle = function(w, h) {
    this.w = w;
    this.h = h;
```

```
}

RectAngle.prototype.area = function() {
    return this.w * this.h;
}
```

然后我们继承RectAngle，实现功能扩展，代码如下所示：

```
ColoredRectAngle = function(color, w, h) {
    RectAngle.call(this, w, h);
    this.c = color;
}
```

下面我们需要将RectAngle中定义的area()函数赋予ColorRectAngle，代码如下所示：

```
ColoredRectAngle.prototype = new RectAngle();
```

上述代码中首先创建一个RectAngle的实例，然后将这个实例设置给ColoredRectAngle的prototype属性，这样ColoredRectAngle的实例就可以顺藤摸瓜，沿着prototype获得RectAngle.prototype中的area()函数，从而实现了继承。

因为我们在创建RectAngle的实例时没有传入任何参数，所以在创建的RectAngle对象实例中会多出两个值为空的变量，w和h。很显然在继承过程中这两个参数是冗余的，我们需要手工删除这两个变量才能完成整个继承过程，如以下代码所示：

```
delete ColoredRectAngle.prototype.w;
delete ColoredRectAngle.prototype.h;
```

至此为止，还需要解决一个问题，ColoredRectAngle的constructor指向是错误的。

如果没有使用ColoredRectAngle.prototpe = new RectAngle();这一步操作，ColoredRectAngle.prototype的指向应该是JavaScript自动创建出的prototype，这个prototype中包含了一个指向ColoredRectAngle的constructor。

但是我们在创建ColoredRectAngle之后执行了ColoredRectAngle.prototype = new RectAngle();。如果现在来访问ColoredRect.prototype.constructor，那么根据自动查找机制，就会找到RectAngle实例中的prototype.constructor，也就是RectAngle。这对于ColoredRectAngle本身是不合时宜的，如果我们运行如下代码：

```
Var coloredRectAngle = new ColoredRectAngle('red', 10, 10);
Alert(coloredRectAngle.constructor);
```

得到的却是父类RectAngle的构造函数，这就导致ColoredRectAngle的对象实例无法正确使用自己的构造函数，因此需要手工把ColoredRectAngle.prototype设置成正确的引用，于是最后需要调用如下代码：

```
ColoredRectAngle.prototype.constructor = ColoredRectAngle;
```

至此，我们完成了完整的JavaScript继承功能。

下面可以来分析一下EXT中是如何实现上述的继承功能的，Ext.extend()的完整代码如下所示：

```
    extend : function(){
        // inline overrides
```

```
    var io = function(o){
        for(var m in o){
            this[m] = o[m];
        }
    };
    var oc = Object.prototype.constructor;

    return function(sb, sp, overrides){
        if(Ext.isObject(sp)){
            overrides = sp;
            sp = sb;
            sb = overrides.constructor != oc ? overrides.constructor : function()
            {sp.apply(this, arguments);};
        }
        var F = function(){},
            sbp,
            spp = sp.prototype;

        F.prototype = spp;
        sbp = sb.prototype = new F();
        sbp.constructor=sb;
        sb.superclass=spp;
        if(spp.constructor == oc){
            spp.constructor=sp;
        }
        sb.override = function(o){
            Ext.override(sb, o);
        };
        sbp.superclass = sbp.supr = (function(){
            return spp;
        });
        sbp.override = io;
        Ext.override(sb, overrides);
        sb.extend = function(o){Ext.extend(sb, o);};
        return sb;
    };
}(),
```

首先这是一个自动执行的函数，它会在EXT被加载的同时执行，在其执行过程中会使用闭包构造出继承所需的局部变量，比如var io用来实现对象之间的属性复制功能，而oc用来保存JavaScript中默认父类Object的构造函数。

Ext.extend()函数实际上支持3个参数。sb表示等待继承的子类，sp表示超类，overrides继承过程中需要覆盖的属性与函数。

不过我们也经常在EXT的源代码中看到对Ext.extend()只使用两个参数的调用方式，这时就需要进行一个转换步骤：

```
if(Ext.isObject(sp)){
  overrides = sp;
  sp = sb;
  sb = overrides.constructor != oc ? overrides.constructor : function(){sp.apply(this, arguments);};
}
```

当只使用两个参数时，这里的if()判断就会为true并执行转换功能。首先将sp也就是第二个参数赋值给overrides，然后将sb也就是第一个参数赋值给sp，最后判断overrides中是否设置了constructor。如果设置了，就用这个属性值作为子类的构造函数，否则使用自动创建的构造函数，自动创建的构造函数会将传入的参数值复制到当前对象中。

下面要实现从父类的prototype中复制属性到子类中，EXT中使用的实现代码如下所示：

```
var F = function(){},
  sbp,
  spp = sp.prototype;

F.prototype = spp;
sbp = sb.prototype = new F();
sbp.constructor=sb;
```

首先定义了空的函数F，然后将超类sp的prototype赋值给spp。

下一步将spp，即超类sp的prototype赋值给F.prototype，这样F就拥有了超类的prototype。

然后执行new F()生成对应的实例，因为F本身是空的，所以在创建实例时不会产生任何额外冗余的变量，这也就省去了上面我们删除无用属性变量的步骤，而F的prototype属性来自于超类，所以不会影响整个继承过程。

在sb.prototype承接了F创建出的实例对象之后，也就完成了子类对父类的继承，最后使用sbp.constructor = sb;将子类的构造函数重新指向到子类即可。

剩下的代码都用来为生成的子类设置一些附加的功能函数，对继承功能不再有影响了。自此，我们便获得了通用的JavaScript继承功能，之后只需要直接使用即可。比如上面演示的普通的JavaScript继承实例可以写成如下形式：

```
ColoredRectAngle = Ext.extend(RectAngle, {});
```

15.2.2 了解 Component 的生命周期

EXT中的所有组件都是继承自Ext.Component，所以我们需要掌握Ext.Component的几个重要生命周期才能准确找到扩展点，实现对这类组件的扩展。

当我们创建一个Ext.Component组件时，会按照以下顺序对组件进行初始化。

- 调用Ext.Apply()复制参数。
- 调用addEvents()添加事件。
- 调用Ext.ComponentMgr.register(this)注册当前组件。
- 调用initComponent()初始化组件。
- 调用initPlugin()初始化插件。
- 调用initState()初始化状态。
- 调用applyToMarkup()或render()进行组件渲染。

这里的initComponent()就留给我们的扩展点。一般情况下在继承了基于Ext.Component组件之后，对组件的初始化操作都可以直接写在initComponent()里，它会在组件初始化前被

自动调用。如果希望修改其他功能，也可以根据上述介绍的功能函数进行修改。

因此，当为Ext.Component及其子类编写自定义扩展时，就需要遵守上述Component的生命周期，在对应的初始化状态重写扩展功能。最常见的操作是覆盖initComponent()函数进行组件初始化，而不是直接重写constructor进行组件的初始化工作。

下面我们会依照Ext.grid.GridPanel扩展出一个更易用的表格控件，期间就会利用到上面讨论到的组件生命周期。

15.3 编写自定义用户扩展

我们的目标是使用最少的代码实现一个功能完整的表格控件，可以想象一下使用通常的方式创建一个GridPanel至少需要哪些步骤。

- 首先设置store获得数据。
- 然后设置columns设置表格中显示的数据。
- 最后加载数据显示表格。

现在我们希望将这3个步骤合并为统一的配置，代码如下所示：

```
var grid = new Ext.ux.MainGrid({
    data: data,
    fields: ['name', 'value'],
    renderTo: 'container'
});
```

为了实现这种效果，我们要扩展Ext.grid.GridPanel实现自定义组件，让自定义组件可以通过上述配置的最小配置生成完整的GridPanel所需要的全部配置，如下面代码所示：

```
Ext.ux.MainGrid = Ext.extend(Ext.grid.GridPanel, {
    initComponent: function() {

        this.autoHeight = true;

        this.tbar = [{
            text: 'create'
        }, {
            text: 'edit'
        }, {
            text: 'remove'
        }];

        this.store = new Ext.data.ArrayStore({
            autoLoad: true,
            proxy: new Ext.data.MemoryProxy(this.data),
            fields: this.fields
        });

        this.columns = [];
        for (var i = 0; i < this.fields.length; i++) {
            this.columns.push({
                header: this.fields[i],
```

15

```
                dataIndex: this.fields[i]
            });
        }

        Ext.ux.MainGrid.superclass.initComponent.call(this);
    }
});
```

从代码中可知，我们仅仅重写了initComponent()函数，在组件初始化之前对必须的参数设置，通过上段代码中设置的data和fields两个参数，为表格生成了store和columns参数，并将store设置为autoLoad:true，使其可以自动加载数据，最终我们可以直接创建MainGrid，将我们扩展的表格显示在页面上。

现在我们知道编写自定义用户扩展组件是一件多么简单多么惬意的事情了。我们需要任何功能都可以直接通过扩展已有的组件实现出来，而且这些功能已经封装在自定义组件中，其后我们就可以直接调用它们，无需重复这些工作了。

15.4 介绍 EXT 的插件体系

在EXT中，用户扩展（User Extension）和插件（Plugin）两种机制为我们提供了对原有功能的扩充途径。与用户扩展不同的时，插件提供了更加灵活的功能扩展方式，有效地利用插件更利于把功能封装成一个一个的模块。

我们重新审视一下Ext.Component组件初始化时调用的函数，如下所述。

- 调用Ext.Apply()复制参数。
- 调用addEvents()添加事件。
- 调用Ext.ComponentMgr.register(this)注册当前组件。
- 调用initComponent()初始化组件。
- 调用initPlugin()初始化插件。
- 调用initState()初始化状态。
- 调用applyToMarkup()或render()进行组件渲染。

可以看到其中有一项initPlugin()，这一步会判断当前组件是否设置了插件。如果存在插件就会对这些插件进行初始化操作，一般都是直接调用插件的init()函数，将插件的某些功能函数绑定在当前组件的一些事件中。当特定事件被触发时就会激活相应的监听函数，从而实现特定的功能。

下面是Ext.Component中用于初始化插件的代码：

```
initPlugin : function(p){
    if(p.ptype && !Ext.isFunction(p.init)){
        p = Ext.ComponentMgr.createPlugin(p);
    }else if(Ext.isString(p)){
        p = Ext.ComponentMgr.createPlugin({
            ptype: p
        });
    }
    p.init(this);
```

```
        return p;
    },
```

从上述代码中可以看到，所谓的插件其实只是一个必须拥有init()函数的对象而已。它没有必要继承什么超类，对其结构也没有什么特殊的要求，只是要求对象中包含一个init()就行了。

假设我们希望为GridPanel添加一个可修改每页显示记录数的扩展，实例代码如下所示：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>ux</title>
    <link rel="stylesheet" type="text/css" href="../../resources/css/ext-all.css" />
    <script type="text/javascript" src="../../adapter/ext/ext-base.js"></script>
    <script type="text/javascript" src="../../ext-all-debug.js"></script>
    <link rel="stylesheet" type="text/css" href="../shared/examples.css" />

    <script type="text/javascript" src="../ux/PagingMemoryProxy.js"></script>
    <script type="text/javascript" src="PageSizePlugin.js"></script>
    <script type="text/javascript">
Ext.onReady(function() {
    var data = [
        ['name1', 'value1'],
        ['name2', 'value2'],
        ['name3', 'value3'],
        ['name4', 'value4'],
        ['name5', 'value5'],
        ['name6', 'value6'],
        ['name7', 'value7'],
        ['name8', 'value8'],
        ['name9', 'value9'],
        ['name10', 'value10'],
        ['name11', 'value11'],
        ['name12', 'value12'],
        ['name13', 'value13'],
        ['name14', 'value14'],
        ['name15', 'value15'],
        ['name16', 'value16'],
        ['name17', 'value17'],
        ['name18', 'value18'],
        ['name19', 'value19'],
        ['name20', 'value20']
    ];

    var store = new Ext.data.ArrayStore({
        proxy: new Ext.ux.data.PagingMemoryProxy(data),
        fields: ['name', 'value']
    });

    var grid = new Ext.grid.GridPanel({
        autoHeight: true,
        store: store,
```

```
            columns: [{
                header: 'name',
                dataIndex: 'name'
            }, {
                header: 'value',
                dataIndex: 'value'
            }],
            bbar: new Ext.PagingToolbar({
                store: store,
                pageSize: 10,
                plugins: new Ext.ux.PageSizePlugin()
            }),
            renderTo: 'container'
        });

        store.load({params: {start: 0, limit: 10}});
    });
        </script>
      </head>
      <body>
        <script type="text/javascript" src="../shared/examples.js"></script>
        <div id="container"></div>
      </body>
    </html>
```

需要注意的是bbar中的plugins参数，我们为其设置了new Ext.ux.PageSizePlugin()，这样在实际使用时，init()函数将自身的onInitView()函数绑定到PagingToolbar的render事件中。在PagingToolbar渲染完毕后，就会触发render事件，onInitView()会被用来向分页工具条中添加选择分页数目的下拉框。

PageSizePlugin的构造函数中会预先创建好一个下拉框ComboBox，并为其定义好一系列可供选择分页数目，默认为10、20、30、50和100。

当用户使用下拉框ComboBox来选择分页数目时，就会触发对应的select事件，与之对应的onPageSizeChanged()函数会被调用。它将通过ComboBox获得用户选择的分页数目，刷新PagingToolbar的pageSize属性。最后调用doLoad()函数重新加载PagingToolbar和当前表格的数据。

最终页面效果如图15-3所示。

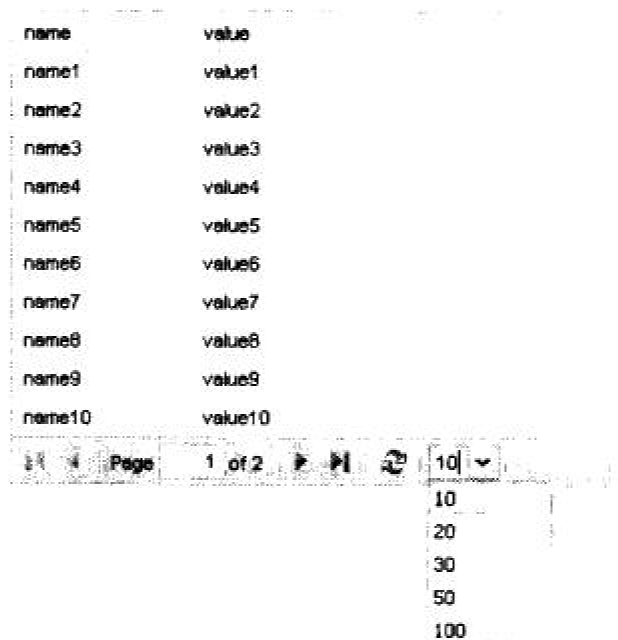


图15-3　可配置列表每页显示数目的插件

插件的价值在于无需更改原有组件，完全通过模块化的方式为原有组件提供额外的功能。从这方面看，用户扩展拥有更强的封装性，使用时可以迅速创建出封装完全的既有功能，而插件则更注重通用性。可以看到一个分页插件完全可以组装在任何一个表格中，无论这个表格是普通GridPanel还是EditorGridPanel，甚至是GroupingGridPanel

都可以使用同样的插件，这对于那些希望将系统设计成完全模块化的人们无疑是一个很好的消息。

实际中是选择用户扩展还是插件，我们还需要根据项目情况具体抉择。

15.5 常用扩展组件（一）UploadDialog

UploadDialog是一套专门用于批量上传文件的扩展组件。对HTML来说，文件上传组件是一个让人挠头的工作。因为出于安全方面的考虑，我们既不能改变文件上传组件的样式，也无法直接为文件上传组件赋值，这就使得我们不得不在被EXT渲染的十分漂亮的页面上放置一个相当丑陋的文件上传组件。

解决这个问题的常见方法是先创建一个原生的文件上传组件，然后把它隐藏起来。当用户点击某个按钮时，模拟点击被隐藏的文件上传组件，以此种方式实现文件上传功能。这种方式对单个文件的上传已经显得过于复杂了，更别提有些情况下我们还希望动态选择多个文件上传。

UploadDialog就为我们提供了一种批量上传文件的途径，通过它可以动态选择多个文件进行批量上传，而且它的界面也相当漂亮，不会出现粗糙的原生文件上传组件。UploadDialog扩展组件的主界面如图15-4所示。

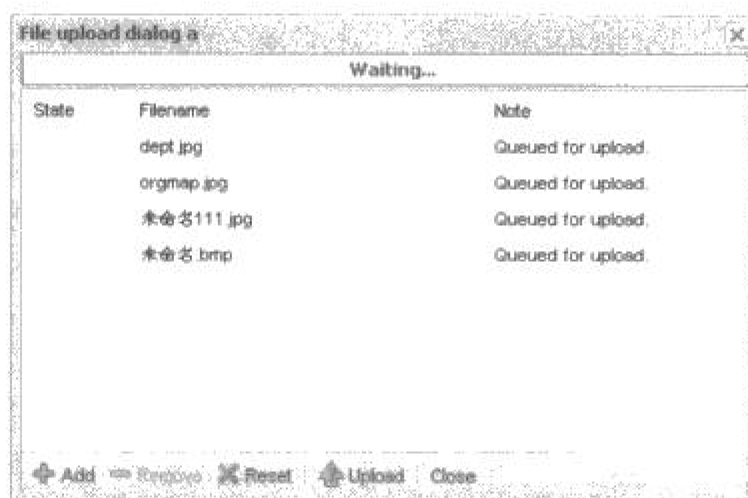


图15-4 UploadDialog扩展组件的主界面

从图15-4中可以看到，通过UploadDialog扩展组件我们可以批量添加多个文件，这些等待上传的文件将以表格形式显示在表格中。创建UploadDialog所需使用的代码如下所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title></title>
    <link rel="stylesheet" type="text/css" href="../../resources/css/ext-all.css">
    <script type="text/javascript" src="../../adapter/ext/ext-base.js"></script>
    <script type="text/javascript" src="../../ext-all.js"></script>
    <script type="text/javascript" src="UploadDialog/UploadDialog.js"></script>
    <link rel="stylesheet" type="text/css" href="UploadDialog/css/Ext.ux.UploadDialog.css">
```

15

```
    <script type="text/javascript">
Ext.onReady(function(){
    Ext.QuickTips.init() ;

    var dialog = new Ext.ux.UploadDialog.Dialog({
        url: '05_01.jsp',
        reset_on_hide: false,
        allow_close_on_upload: true,
        upload_autostart: false
    });

    dialog.show('show-button');
}) ;
    </script>
  </head>
  <body>
    <button id='show-button'>show</button>
  </body>
</html>
```

从上面代码可以看出，我们在页面中引入对应的JavaScript脚本与CSS样式后，直接创建了一个UploadDialog对象，并设置了上传使用了url参数。这样在我们点击上传按钮时就会将选中的文件一次上传到'05_01.jsp'，其他参数：reset_on_hide会在窗口隐藏时清空表格中的文件，allow_close_on_upload会在关闭窗口时进行上传，upload_autostart设置为false就不会在用户选择一个文件后直接进行上传操作。

我们可以对这些等待上传的文件进行添加、删除、清空等操作。在确定哪些文件将要上传后，可以直接点击Upload按钮上传，这些文件会依次上传到服务端，可以通过UploadDialog上端的进度条观察到上传的进度情况。点击上传后效果如图15-5所示。

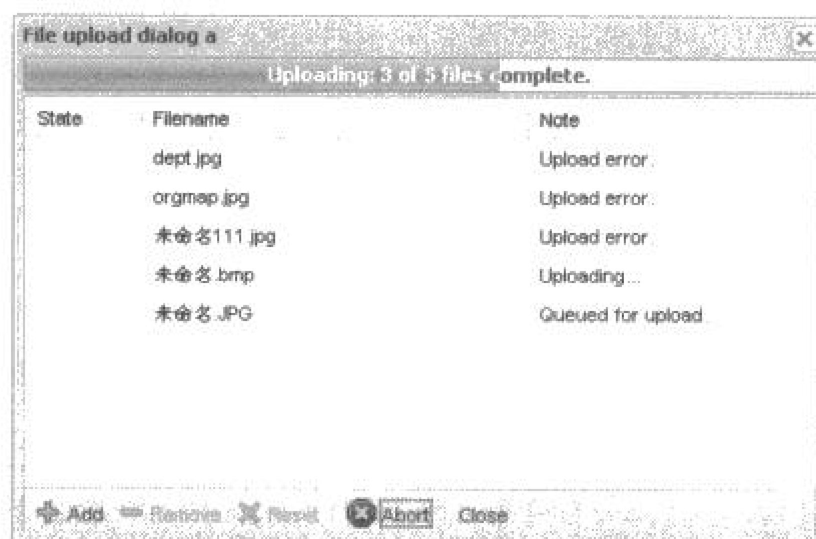


图15-5 上传过程中的界面效果

图15-5中可以看到点击上传按钮后对整个进度的监控状况，UploadDialog中不只列出了整体的上传进度，还显示了每个文件的上传状况。图15-5中显示正在上传第四个文件，而且前三个文件都上传失败了。

文件上传成功后界面显示效果如图15-6所示。

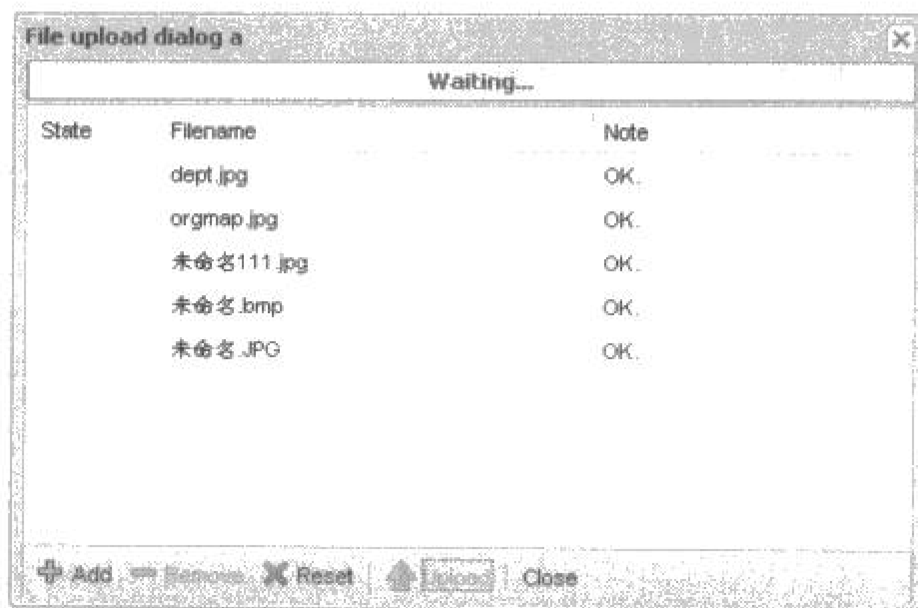


图15-6 文件上传成功后的界面效果

UploadDialog.js的源代码相当复杂，这从文件大小就可以初见端倪。UploadDialog.js本身就有41 KB，其中实现了事件队列（EventQueue）、有限状态机（FSA）、浏览按钮（BrowseButton）、文件记录（FileRecord），并最终将这些组件组装为我们上面使用的UploadDialog。

UploadDialog直接继承自`Ext.Window`，整体上就是一个弹出窗口，其中包含了上方的进度条，中间的表格和最下面的功能按钮。为了实现上传提交功能，UploadDialog中还自动创建了对应的上传表单。在点击BrowseButton时，就会自动生成对应的文件上传组件，并将文件上传组件插入上传表单中。这些操作都是由UploadDialog自动维护的，无需使用者过多关心具体实现的每一步骤。

UploadDialog的官方网址为http://max-bazhenov.com/dev/upload-dialog-2.0/index.php，官方的下载包还提供了整体组件的处理图形介绍，感兴趣的读者可以去看一下。

15.6 常用扩展组件（二）ManagedIFrame

某些情况下，我们需要在EXT中引用其他页面中的内容，这时可以使用`Ext.Updater`通过Ajax加载页面内容，也可以直接使用`iframe`的形式，在`iframe`中加载其他页面。这是因为在使用`Ext.Updater`的情况下，加载的HTML片段是直接插入到当前页面中，在这一过程中有可能引发各种冲突，尤其在加载复杂页面内容的时候更容易出现这种问题。而使用iframe就可以避免这些问题，因为iframe会将内部页面与外部完全隔离。iframe内部可以看作一种独立的窗口，无论内部加载什么样的内容都不会对外部页面造成任何影响，所以大可放心使用。

但是也因为iframe这种隔离特性导致了不小的问题，比如我们很难直接控制iframe内部页面的加载使用。如果我们希望从外部页面调用iframe中加载的页面对象也需要费些周折。最主要的问题是无法让iframe中的页面像其他部分一样直接进行布局。

`ManagedIFrame`就是为了解决这一系列问题而被创造的，`ManagedIFrame`会自动在内部创

建一个iframe对象，并对iframe中的事件和样式都进行封装，向外部提供统一的调用接口。这让我们可以通过它更便捷地处理iframe，比如ManagedIFrame中提供的ManagedIFrame.Component和ManagedIFrame.Panel，使用它们与使用普通的Ext.Component和Ext.Panel一样，它们提供了与之前完全相同的方式来操作这些内部嵌入的iframe。

使用ManagedIFrame的代码如下所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title></title>
    <link rel="stylesheet" type="text/css" href="../../resources/css/ext-all.css">
    <script type="text/javascript" src="../../adapter/ext/ext-base.js"></script>
    <script type="text/javascript" src="../../ext-all.js"></script>
    <script type="text/javascript" src="ManagedIFrame/uxvismode.js"></script>
    <script type="text/javascript" src="ManagedIFrame/multidom.js"></script>
    <script type="text/javascript" src="ManagedIFrame/mif.js"></script>
    <script type="text/javascript" src="ManagedIFrame/mifmsg.js"></script>
    <script type="text/javascript">
Ext.onReady(function(){
    Ext.QuickTips.init() ;

    new Ext.ux.ManagedIFrame.Panel({
        title: 'iframe',
        defaultSrc: '06-01-01.html',
        loadMask: true
    }).render(document.body);
}) ;
    </script>
  </head>
  <body>
  </body>
</html>
```

上述代码中创建了包含iframe的Panel组件，它和我们之前使用的Ext.Panel组件完全相同，可以为它设置title和loadMask等参数。唯一不同的是在ManagedIFrame.Panel中包含了一个iframe，iframe中实现的内容来自06-01-01.html页面。

上述代码的显示效果如图15-7所示。

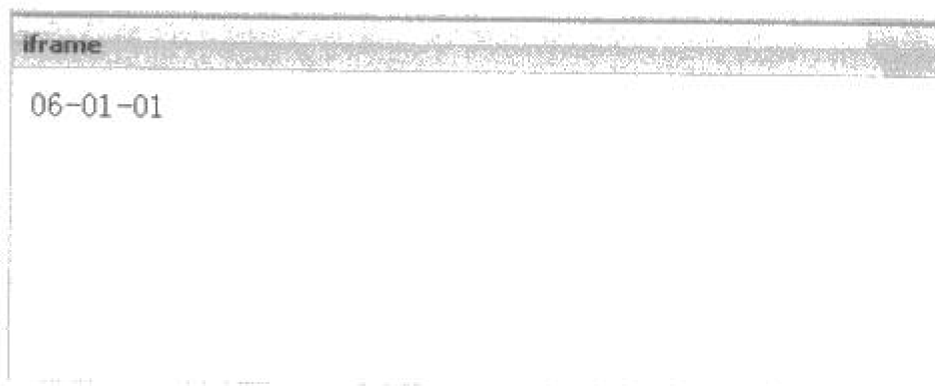


图15-7　使用ManagedIFrame显示的效果

如果不需要使用Ext.Panel中提供的标题、工具条等额外功能，也可以直接使用ManagedIFrame.Component。它继承自Ext.BoxComponent，足以提供最基本的布局功能。

使用ManagedIFrame.Component的实例代码如下所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title></title>
    <link rel="stylesheet" type="text/css" href="../../resources/css/ext-all.css">
    <script type="text/javascript" src="../../adapter/ext/ext-base.js"></script>
    <script type="text/javascript" src="../../ext-all.js"></script>
    <script type="text/javascript" src="ManagedIFrame/uxvismode.js"></script>
    <script type="text/javascript" src="ManagedIFrame/multidom.js"></script>
    <script type="text/javascript" src="ManagedIFrame/mif.js"></script>
    <script type="text/javascript" src="ManagedIFrame/mifmsg.js"></script>
    <script type="text/javascript">
Ext.onReady(function(){
    Ext.QuickTips.init() ;

    var mc = new Ext.ux.ManagedIFrame.Component({
        defaultSrc: '06-01-01.html'
    }).render('mc');

    Ext.get('b').on('click', function() {
        mc.setLocation('06-02-01.html');
    });

}) ;
    </script>
  </head>
  <body>
    <div id="mc"></div>
    <button id="b">change</button>
  </body>
</html>
```

ManagedIFrame.Component可以看作简易版的Panel，它完全不包含标题工具条等附属组件。我们在页面上看到的只是平面内容，只不过这些内容都来自外部页面06-02-01.html。我们还可以直接通过ManagedIFrame.Component的setLocation()函数修改iframe中显示的页面，一切操作都是非常的方便。

使用ManagedIFrameComponent的显示效果如图15-8所示。

除了以上介绍的ManagedIFrame.Panel和ManagedIFrame.Component，mif.js中还提供了ManagedIFrame.Portlet和ManagedIframe.Window等组件，分别用以对Portal和弹出窗口进行支持。其实现原理都是继承对应的父类，然后在显示主体部分内嵌一个iframe。我们可以根据实际开发时的需要进行选择。

ManagedIFrame.Window 的显示效果如图15-9所示。



图15-8 ManagedIFrame.Component的显示效果

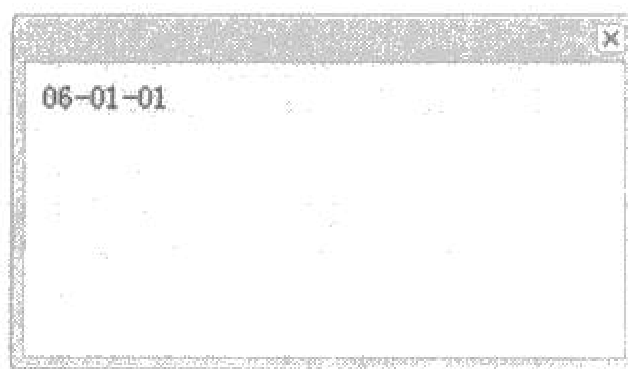


图15-9 ManagedIFrame.Window的显示效果

mif.js是一个更加庞大的组件体系，单是mif.js的源文件就有125KB。它主要的功能是对iframe中的事件进行封装，以此保证外部组件可以用EXT中支持的方式对事件进行监听和操作。在对iframe的封装完成之后，又在iframe的基础之上构造了常用的Component、Panel、Portlet、Window等常用功能组件，我们可以根据实际情况选择适合的组件进行应用。

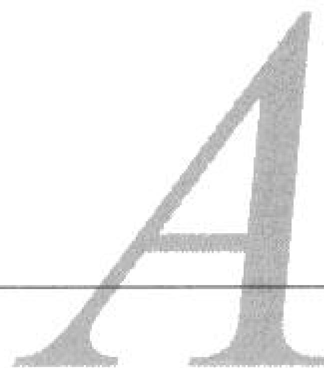
15.7 小结

本章主要介绍了EXT提供的用户扩展与插件机制。编写用户扩展或插件，可以尽可能的实现代码复用，减少代码重复的数量。

本章的后面部分介绍了两个实际开发中常用的复杂扩展组件，它们分别是用于文件批量上传的UploadDialog和可以更容易调用外部页面但更易管理的ManagedIFrame。通过它们两个我们可以看到利用EXT的扩展功能可以实现多么强大的功能，只要用心我们也可以实现如此功能丰富的扩展组件。

附录 A

EXT常见问题



本章内容

- EXT到底是收费的还是免费的
- 弹出对话框不宜单独使用
- 日期选择控件不宜单独使用
- 如何查看EXT中的API文档
- 如何在页面中引用EXT
- EXT的汉化
- 使用Ajax获取和提交数据时出现乱码
- 如何执行`autoLoad`加载的页面中的JavaScript脚本
- 有关表格的一些小问题
- 有关树形的一些小问题
- 如何使用`input type="image"`
- `Ext.Window`中的`closeAction`
- 使用同步Ajax

A.1 EXT 到底是收费的还是免费的

很多读者对这个问题感兴趣，实际上Jack已经把答案写在http://www.extjs.com/license里了，对EXT的授权形式做了详细的说明。

EXT的授权形式有3种，如下所示。

(1) 免费授权

大家先别高兴，免费协议是有限制的，是不能随意使用的。只有满足下列条件之一，才能获得免费授权。

- 如果你在做一个开源项目，而且这个项目里没有使用任何非开源软件，那么可以免费使用EXT。
- 如果你是用于自己学习研究或教学等非营利性目的，那么可以免费用EXT。
- 如果你不愿意向EXT开发团队提供资金上的资助，还是想要把EXT用在自己的商业项目中，那么也可以使用EXT，但是你不能将EXT用作为软件开发库，也不能将EXT用作开发工具。

是不是很复杂？简单来说就是，如果你将EXT用于非营利性目的，就可以在LGPL[1]协议下免费用。如果你将EXT用于营利性目的，就不能再把EXT封装起来当工具库卖，除此之外的领域都可以使用EXT。

> **注意** 如果你使用的是EXT 2.1或更高的版本，开源协议便改成GPL协议。

(2) 企业授权

如果你不愿意受到免费协议的限制，如果你们内部协议要求必须用企业授权，如果你愿意在经济上支持EXT开发团队的持续发展，那么可以获得EXT的企业授权。

(3) OEM / Reseller License

你要是想把EXT封装为软件开发库（software development library）或工具包（toolkit）来卖，就需要取得EXT开发团队的专门协作授权，免费授权和普通的企业授权都是不允许客户使用EXT制作开发库和工具包的。

Jack还介绍了购买OEM协议的好处，比如不用受LGPL的限制，你的产品就成了市场上被EXT官方开发团队支持的产品，同时你也获得了更多的合作机会，也获得了EXT团队直接授权的技术支持。

或许这样说还是不够清晰，其实就是说，如果你想开发一套IDE，还是去跟Jack好好谈谈吧。

上面对EXT提供的多种授权方式做了简要的介绍，大家可以选择最适合自己的一种。

A.2 弹出对话框不宜单独使用

在EXT中，不宜把弹出对话框单独拿出来使用。建议大家去看看Lightbox这款JavaScript组件，它本来是一个基于prototypejs开发的弹出窗口控件，现在已经发展出许多由其他底层库支持的分支。更多详细信息请参见：http://www.huddletogether.com/projects/lightbox/。

A.3 日期选择控件不宜单独使用

在EXT中，不宜把日期选择控件单独拿出来使用。JavaScript的日期选择控件很多，这里推荐大家使用jscalendar这款JavaScript组件，它不但支持日期选择，还可以直接将日历显示在页面上。更多详细信息请参见：http://www.dynarch.com/projects/calendar/。

A.4 如何查看 EXT 中的 API 文档

因为在EXT中读取API文档时要使用Ajax，而在本地文件系统中Ajax一直返回失败状态，所以无法正常显示页面，解决方法有两种，如下所示。

- 将整个EXT包复制到IIS或Tomcat这类服务器上，然后通过服务器访问API文档，这样Ajax就可以返回正常结果。

---

① 协议原文请参考 http://www.gnu.org/licenses/lgpl-3.0.txt。LGPL被认为是能够较好地保护开发者利益的一个开源协议。简单来说，遵守LGPL协议的软件可以使用，但不能修改。如果要改，必须把修改部分的代码公开。

- EXT官方网站上有人发布过localXHR.js扩展插件，只要在HTML中引入这个插件就可以让EXT支持在本地文件系统中使用Ajax。找到localXHR.js文件，将它复制到docs目录下，然后在index.html中加入`<script Src="localXHR.js"></script>`语句。

注意　这一行要加在ext-all.js的后面，然后直接打开index.html就可以阅读API文档了。

A.5　如何在页面中引用 EXT

EXT只是单纯的JavaScript，引用方式和使用一般的外部JavaScript文件相同，如下面的代码所示：

```
<script type="text/javascript" src="../../adapter/ext/ext-base.js"></script>
<script type="text/javascript" src="../../ext-all.js"></script>
```

如果你想把EXT放入自己开发的项目中，需要包括以下几部分:ext-all.js、adapter目录和resources目录。

(1) ext-all.js是对所有源文件压缩合并后的结果，包含了所有EXT控件。

开发时可以考虑使用ext-all-debug.js，这是未经压缩的版本，通过Firebug可以找到具体是哪行出现了问题。如果使用IE，也可以用其中附加的`Ext.log`进行调试，但功能没有Firebug强。

(2) adapter目录下是各种适配器，选择其中一种即可。

目前提供的有ext-base.js、Prototypejs、YUI和jQuery。EXT在这些核心库的基础上构筑起强大的功能，开发者根据自己的实际需要选择对应的适配器，便可以在之前的经验基础上进行EXT开发。

(3) resources目录下是CSS样式和图片资源，它们不一定和JavaScript脚本放在一起，只要CSS资源和图片的位置对应即可。

在使用EXT的样式和图片时，只需要在页面中引入ext-all.css，如下面的代码所示：

```
<link rel="stylesheet" type="text/css"
href="../../resources/css/ext-all.css" />
```

(4) 如果需要国际化支持，还需要从build目录下复制locale目录，导入到项目中，下面会详细介绍。

A.5.1　常见的“EXT is not defined”错误

如果在HTML页面中没有按照首先引入ext-base.js，然后再引入ext-all.js的顺序，就会出现“Ext is not defined”的错误。它的意思是说EXT这个对象还没有创建，我们不能直接使用未定义的对象。因为EXT这个作为顶级命名空间的对象是在适配器（adapter）中定义的，无论我们选择ext-base、Prototypejs、jQuery还是YUI作为适配器，都必须把对应的适配器脚本放在ext-all.js前面。如果误将ext-all.js放在适配器脚本的前面，就会出现“EXT is not defined”的错误。

同样，如果你需要在页面中引入自定义的EXT扩展组件，就必须把自己编写的JavaScript脚本放在适配器脚本和ext-all.js的后面，否则就会出现“xxx is not a constructor”（xxx不是构造函数）

的错误。由此可见，JavaScript脚本的引入顺序是很重要的。

A.5.2　IE 逗号问题

在IE浏览器中不能出现多余的逗号，在数组的最后一列或最后一个属性中也不能出现多余的逗号，否则会出现语法错误。

```
var array = [
    'name1', 'name2', 'name3', // IE下会报错
];
```

而Firefox环境下却没有这个问题，它会自动忽略结尾部分多余的逗号。在IE中如果出现了多余的逗号，整个页面都可能无法正常显示。所以，如果在Firefox中显示正常，但在IE中却不能正常显示，很有可能是因为多了一个逗号。

A.6　EXT 的汉化

EXT提供了国际化的脚本，这些脚本都在build/locale/目录下，你只需要把对应语言的脚本加入页面就可以了，比如我们需要使用ext-lang-zh.CN.js对EXT进行汉化，如下面的代码所示：

```
<script type="text/javascript" src="../../adapter/ext/ext-base.js"></script>
<script type="text/javascript" src="../../ext-all.js"></script>
<script type="text/javascript"
src="../../build/locale/ext-lang-zh_CN.js"></script>
```

注意　把翻译的脚本放在ext-all.js之后，翻译脚本文件的编码格式是UTF-8。如果你的应用需要GB-2312或其他编码格式，请把国际化文件转换为你所需要的编码格式。

A.7　使用 Ajax 获取和提交数据时出现乱码

使用英文时是不会出现乱码的，只有使用中文时才有可能出现乱码。在Windows操作系统中，保存文件的默认编码是GB-2312，而Ajax传输数据的默认编码是UTF-8。因此，我们建议大家将数据编码格式统一为UTF-8，不但可以解决眼前的乱码问题，而且对以后扩展为多语言也有好处。

A.8　如何执行 `autoLoad` 加载的页面中的 JavaScript 脚本

可以在TabPanel中使用scripts:true属性，它可以让TabPanel加载页面中的JavaScript脚本，如下面的代码所示：

```
var tabItem = tabPanel.add({
    id:title,
    title:title,
    closable:true,
    autoScroll:true,
    autoLoad:{url:url,scripts:true}
});
tabPanel.activate(tabItem);
```

A.9　有关表格的一些小问题

本节将列出一些在使用表格控件时常见的问题，以及对应的解决方法。

A.9.1　如何让表格所有的列都支持排序

当然，你可以在每个列上都设置sortable: true属性，让表格所有的列都支持排序功能。我们也可以修改ColumnModel的属性defaultSortable，默认情况下它是false。如果希望所有列在默认情况下都可以进行排序，就在构造ColumnModel之后设置cm.defaultSortable = true。这样，这个ColumnModel下的所有列在默认情况下都是可排序的。

A.9.2　修改表格的 ColumnModel 和 Store

cm和ds决定了表格的显示方式和显示数据。有人提出希望用一个表格显示不同形式的数据，这些数据的列模型与数据不一定相同。这时我们可以使用reconfigure()函数，它的第一个参数是Ext.data.Store，第二个参数是Ext.grid.ColumnModel。只需要把新的ds和cm传入，就可以让表格显示新的数据了。

A.9.3　动态为 ds 添加参数 baseParams

关于如何给表格设置查询参数，我们可以把参数写到ds设置的url里。虽然没有灵活性可言，但也算是一种解决方法。

还有一种方法是使用ds中的baseParams，你为它设置的参数会在reload时自动加到url后面。如果每次都重新设置，就可以用ds.baseParams = {name: 'value'}。如果只是修改其中的一项，那么最好还是用ds.baseParams['name'] = 'value'的形式，以免把其他参数都清空了。

用baseParams的最大好处就是可以保留参数，下一次即使直接调用reload()参数也不会丢失。如果你只是把参数加入到load({params:{name:'value'}})中，那么下次单击翻页按钮时，上次设置的参数就不存在了。

A.10　有关树形的一些小问题

本节将列出一些在使用树形控件时常见的问题，以及对应的解决方法。

A.10.1　如何选中树中的某个节点

- 每个TreeNode都有select()函数，调用这个函数就是选中节点。
- 使用TreePanel的selectPath()函数，传入的参数是想要选中的节点的path值，你可以使用node.getPath()获得这个值。例如，如果根节点的id是root，那么就使用selectPath('/root');；如果根节点下有一个id为leaf的节点，我们要选中这个节点就要使用selectPaht('/root/leaf');，依此类推。

- 通过SelectionModel选择节点，treePanel.getSelectionModel()便可以获得属性的选择模型。sm有一个select()函数，传入TreeNode就可以选中这个节点了。它还提供selectPrevious()和selectNext()方法，选中当前节点的上一个或下一个节点。

A.10.2　刷新树的所有节点

TreeNode有一个reload()函数，会刷新它下面的所有节点。如果想刷新整棵树，就要取得根节点rootNode，然后调用reload()。

A.10.3　如何取得 JSON 中自定义的属性

可以通过treeNode.attributes取得JSON自定义的属性。如果在JSON中设置了{id:1, text:'text',key:'key'}，在获得对应节点的treeNode后，使用treeNode.attributes.key的方式获得key对应的值。

A.11　如何使用 input type="image"

关于input type="password"、input type="submit"、input type="file"和input type="image"，在EXT中没有提供对应的控件，我们可以根据需要使用inputType对Ext.form.TextField进行改装。

这里input type="image"比较特殊，因为需要通过src属性指定显示的图片。默认情况下Ext.form.TextField生成的DOM没有这个属性，我们需要修改autoCreate参数，如下面的代码所示：

```
{
    fieldLabel: '证件照片',
    name: 'smallimg',
    inputType: 'image',
    autoCreate: {tag: "input", type: "image", src:'http://… …/noavatar.gif' },
    width: 120,
    height: 140
}
```

autoCreate使用的是DomHelper的语法，这样它就会生成一个带有src的DOM了。

A.12　Ext.Window 中的 closeAction

创建Ext.Window时，closeAction的可选值有两个，分别是close和hide。我们一般都用hide，关闭窗口后不会将对象销毁，下次使用时直接调用show()函数就可以让窗口显示出来，这样可以避免重复创建窗口的对象和DOM造成的性能问题。

A.13　使用同步 Ajax

在EXT中，使用Ajax的方式是异步调用，在Ajax请求处理还未完成时，程序就会开始执行后面的代码，所以我们不能根据代码中语句的先后顺序来判断Ajax请求是否成功，只能通过为Ajax

设置监听函数，在请求完成时执行相应的操作。

除了函数名不同，Ext.lib.Ajax.asyncRequest(method, uri, callback, postData)的参数类型和使用方法都与Ext.lib.Ajax.request相同。

如果要实现同步，就一定要conn.open("GET", url, false);，而asyncRequest的代码是conn.open("GET", url, true);，下面是一种实现方法。

```
var a = "{" + params + "}";
var conn = Ext.lib.Ajax.getConnectionObject().conn;
url = url + '?_ajaxP=' + encodeURIComponent(a);
conn.open("GET", url, false);
conn.send(null);
var rs = Ext.decode(conn.responseText);
return rs;
```

附录 B

EXT对AIR的支持

本章内容

- 什么是AIR
- 使用AIR编写Hello World
- 在AIR下使用EXT
- Ext.air.FileProvider
- Ext.air.NativeWindow
- Ext.sql
- Ext.air.Sound
- 一个完整的示例

EXT从2.0.2版本开始支持Adobe-AIR，我们可以通过EXT封装的函数编写AIR应用。

B.1 什么是 AIR

AIR（Adobe Integrated Runtime）是一个跨操作系统的运行环境，同时又是一个虚拟机。可以把以前写在服务器上的应用打包放到它下面运行，它支持HTML、JavaScript、CSS和Flash。

从某方面讲，AIR有如下3点优势。

- 因为应用程序需要用户手动安装，所以应用程序拥有更高的操作权限，例如可以访问本地文件或数据库。
- 应用程序可以与操作系统进行交互，例如操作本地原生窗口，可以把窗口收缩到系统任务栏里，还可以播放本地保存的音乐。
- 安装后的应用程序可以像普通程序一样执行和卸载，可以在系统设置中直接管理。
- 从安装使用上讲，AIR有如下两点不足。
- 需要安装一个11 MB的运行环境，然后才能做其他的事情。
- 每个AIR程序都要先安装才能使用，操作显得很烦琐。

B.2 使用 AIR 编写 Hello World

这是一个简单的示例，具体步骤如下所示。

(1) 下载并安装JRE，下载地址为：http://java.sun.com/javase/downloads/index.jsp。

(2) 下载AIR SDK，下载地址为： http://www.adobe.com/products/air/tools/sdk/。

AIR SDK解压后即可使用，最好设置一下环境变量，把bin目录下的adl.exe和adt.exe加入到path变量中。adl.exe可以用来调试，adt.exe则用来打包发布。

(3) 建立一个名为helloworld的目录，将它作为项目的根目录。

在helloworld目录下新建一个名为application.xml的文件，如下面的代码所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://ns.adobe.com/air/application/1.0">
    <id>examples.html.HelloWorld</id>
    <version>0.1</version>
    <filename>HelloWorld</filename>
    <initialWindow>
        <content>helloworld.html</content>
        <visible>true</visible>
        <width>400</width>
        <height>200</height>
    </initialWindow>
</application>
```

再新建一个helloworld.html文件，它里边只有两个单词“Hello World”。

现在可以测试我们的第一个AIR项目了。在cmd控制台下进入helloworld目录，执行“adl application.xml”命令，就会看到一个宽400（像素），高200（像素），内容是Hello World的窗口，如图B-1所示。

示例在air/01_helloworld/中。



图B-1 第一个AIR程序HelloWorld

B.3 在 AIR 下使用 EXT

因为EXT是基于HTML、CSS和JavaScript的控件库，所以它可以直接在AIR环境下运行。

要在AIR下使用EXT，我们需要为程序进行一些配置，具体步骤如下所示。

(1) 先引入EXT需要的脚本，如下面的代码所示：

```
<link rel="stylesheet" type="text/css" href="ext-2.0.2/resources/css/ext-all.css" />
<script type="text/javascript" src="ext-2.0.2/ext-base.js"></script>
<script type="text/javascript" src="ext-2.0.2/ext-all.js"></script>
<script type="text/javascript" src="ext-2.0.2/ext-lang-zh_CN.js"></script>
```

(2) 添加AIR的JavaScript脚本，这个脚本文件放在air sdk/frameworks目录下，如下面的代码所示：

```
<script type="text/javascript" src="adobe-air/AIRAliases.js"></script>
```

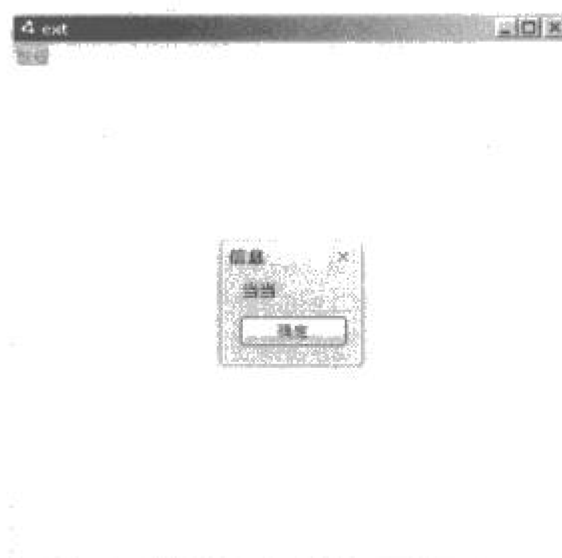
(3) 添加EXT对AIR的封装，这里使用的封装脚本文件来自Simple Task示例，如下面的代码所示：

```
<link rel="stylesheet" type="text/css" href="ext-air/resources/ext-air.css" />
<script type="text/javascript" src="ext-air/ext-air.js"></script>
```

这样就可以在AIR中使用EXT了，如果只想使用EXT，而不需要使用EXT封装好的AIR功能函数，就不需要执行后面的两步了。现在可以测试应用，看看EXT在AIR下的执行结果，如图B-2所示。

示例在air/02_ext/中。

配置好这些后，就可以尝试下面示例中演示的EXT与AIR的封装功能了。



图B-2　在AIR环境下调用EXT的对话框

B.4　Ext.air.FileProvider

通过AIR，EXT就拥有了访问本地文件的权限，我们可以把状态信息保存到本地文件中。

如果希望使用这项功能，需要在Ext.onReady的开始处添加如下代码：

```
Ext.state.Manager.setProvider(new Ext.air.FileProvider({
    file: 'ext.state',
    defaultState: {
        mainWindow: {
            width: 800,
            height: 200,
            x: 10,
            y: 10
        }
    }
}));
```

我们创建一个Ext.air.FileProvider，并把它设置到Ext.state.Manager中，此后就可以用它保存组件的状态信息。

Ext.air.FileProvider里有两个参数，如下所示。

(1) File：表示用来保存状态信息的文件路径，默认是extstate.data。

如果使用相对路径，AIR会把它放到C:\Documents and Settings\Administrator\Application Data\examples.ext.FileProvider\Local Store\ext.state下。

这个路径会根据登录用户的不同而发生改变，其实Windows XP系统会为每个登录的页面创建一个保存用户设置的根目录。如果Windows XP安装在C盘，那么使用Administrator这个账号登录的用户的主目录就在C:\Documents and Settings\Administrator\下，AIR就是在这个目录的基础上，依照下述形式来保存状态信息文件的：

Windows XP登录用户的主目录/Application Data/application.xml中定义的项目id（安装后还会加上后缀）/配置的状态文件名。

我们讨论过，在设置state时，可能会出现的一些莫名其妙的问题，如果希望清空以前保存的状态，可以直接进入对应目录，删除状态文件。

(2) defaultState：用来为第一次启动程序时设置默认状态。

它使用stateId作为标志，后面紧接stateId对应组件的状态设置。

mainWindow有特殊含义，它为air的主窗口配置状态，可以把这看作配置AIR主窗口的一个方式。

示例在air/03_fileprovider/中。

B.5 Ext.air.NativeWindow

它通过AIR调用本地原生窗口，如下面的代码所示：

```
var win = new Ext.air.NativeWindow({
    id: 'mainWindow',
    instance: window.nativeWindow,
    minimizeToTray: true,
    trayIcon: 'ext-air/resources/icons/extlogo16.png',
    trayTip: '原生窗口',
    trayMenu : [{
        text: '打开窗口',
        handler: function(){
               win.activate();
        }
    }, '-', {
        text: '退出',
        handler: function(){
               air.NativeApplication.nativeApplication.exit();
        }
    }]
});
```

简单看一下上面用到的参数，如下所示。

- 为Ext.air.NativeWindow定义一个id，让这个窗口成为我们使用的主窗口。
- instance传入的window.nativeWindow其实就是原生窗口，如果你想直接操作air这个原生窗口实例，可以通过win.instance得到对它的引用。
- 如果希望打开新页面，则可以传递html:'test.html'。
- minimizeToTray会在我们对窗口执行最小化操作时让窗口收缩到任务栏中。

trayIcon是收缩后任务栏上显示的图标，trayTip表示鼠标放到这个任务栏图标上显示的提示信息。trayMenu是在图标上单击鼠标右键时弹出的菜单，这里包含一个“打开窗口”项和一个“退出”项。

示例在air/04_nativewindow/中。

B.6 Ext.sql

AIR里使用的数据库是一款名为sqlite3的嵌入式数据库，EXT在此基础上进行了封装。下面我们来看一些简单的操作。

- var conn = Ext.sql.Connection.getInstance();

这是一个工厂模式，根据Ext.isAir判断是返回AIR还是google gear的实现。

注意 目前市面上发布的EXT所提供的isAir属性都无法识别air-1.0，只有使用simple task示例中的isAir = navigator.userAgent.toLowerCase().indexOf("adobeair") != -1;才能正确识别。如果你不想用它里面未压缩的ext-base.js和ext-all.js，那就需要把这个修改添加到自己的项目中。

- conn.open('ext.db');
连接到数据库，数据库文件是当前目录下的ext.db。如果文件不存在，那么会创建一个新文件。

```
conn.createTable({
name: 'user',
key: 'userId',
fields:[
{name: 'userId', type: 'int'},
    {name: 'name', type: 'string'},
    {name: 'sex', type: 'string'},
    {name: 'descn', type: 'string'}
]
});
```

创建一个名为user的表，主键是userId，fields用来定义字段名称和类型。

```
var userDao = conn.getTable('user','userId');
```

根据表名和表主键获得一个表对应的dao对象。获得dao对象后，可以执行查询和添加等操作，代码如下所示：

```
var users = userDao.select();

var newUser = {
    userId : 'username',
    name : 'kayzhan',
    sex : 'male',
    descn : 'description'
}

userDao.insert(newUser);
```

B.7 Ext.air.Sound

AIR还可以直接播放本地目录下的音乐文件，这是不需要其他插件就能实现的原生支持。在EXT中，对这项功能进行了封装，只需要一行代码便可以播放音乐，如下所示：

```
Ext.air.Sound.play('beep.mp3',0);
```

其中，第一个参数是音乐文件的名称，第二个参数是跳过起点多少毫秒（ms）开始播放。注意，第二个参数不能省略。

示例在air/06_sound/中。

B.8　一个完整的示例



图B-3　在AIR中使用EXT的表格控件实现CRUD功能

使用AIR制作一个有浏览、添加和删除功能的表格（如图B-3所示），数据的读取和更新都使用Ext.sql实现。

下面我们将使用ext-2.0.2和ext-air来实现这些功能。

B.8.1　使用 Ext.data.Store 与数据库进行交互

表格需要通过Ext.data.Store获得数据，以前我们都是通过MemoryProxy和HttpProxy这类组件从内存或HTTP服务器中获得数据。现在我们用AIR，需要从数据库获得数据，所以先要实现一个UserStore，把数据库和表格联系起来。

先定义一个Record，统一数据结构，如下面的代码所示：

```
var User = Ext.data.Record.create([
    {name: 'userId', type: 'int'},
    {name: 'name', type: 'string'},
    {name: 'sex', type: 'string'},
    {name: 'descn', type: 'string'}
]);
```

现在可以为User创建UseStore了，如下面的代码所示：

```
UserStore = Ext.extend(Ext.data.Store, {
    constructor: function(conn){
        UserStore.superclass.constructor.call(this, {
            reader: new Ext.data.JsonReader({
                id: 'userId',
                fields: User
            })
        });
        this.conn = conn;
        this.proxy = new Ext.sql.Proxy(conn, 'user', 'userId', this);

        this.proxy.on('beforeload', this.prepareTable, conn);
    },

    addUser : function(data){
        this.suspendEvents();
        this.clearFilter();
        this.resumeEvents();
        this.loadData([data], true);
        this.fireEvent('datachanged', this);
    },

    prepareTable : function(){
        try{
            this.createTable({
```

```
                    name: 'user',
                    key: 'userId',
                    fields: User.prototype.fields
                });
            }catch(e){console.log(e)}
        }
    });
```

整个store分为3大部分，如下所示。

(1) constructor：构造函数，用于进行初始化。

首先，调用Ext.data.Store的构造函数初始化读取数据的JsonReader。JsonReader的id是数据库表的主键userId，fields部分是字段定义。其次，将Ext.sql.Connection赋值给this.conn。然后，使用conn、数据库表名、表主键名和store作为参数，生成Ext.sql.Proxy实例。

最后监听proxy的beforeload事件，在读取数据之前调用prepareTable函数初始化表结构。如果表已经存在，那么就不会删除表里已有的数据。

(2) addUser：添加一条用户信息。注意，不能是Record，只能是简单的JSON对象。

先调用clearFilter()清空过滤信息，让所有数据都显示出来，然后调用loadData()添加数据，最后触发datachanged事件。

(3) prepareTable：创建表结构。

调用conn的createTable函数，创建表名为user、主键为userId、字段定义为User.prototype.fields的表。

这里解释一下，在构造函数中创建Ext.sql.Proxy时会将store本身作为参数传递给Proxy，Proxy则会对store的add、update和remove这3个事件进行注册，保证可以在用户操作store时对数据库进行同步操作。

```
this.store.on('add', this.onAdd, this);
this.store.on('update', this.onUpdate, this);
this.store.on('remove', this.onRemove, this);
```

如果不希望Proxy将store中的操作同步到数据库，只需要在创建Proxy时加上readOnly参数，如果为true，Proxy将不会自动同步数据。

```
this.proxy = new Ext.sql.Proxy(conn, 'user', 'userId', this, true);
```

B.8.2　打开数据库连接并创建 store

如之前提到的，我们先连接数据库，然后创建store。

```
var conn = Ext.sql.Connection.getInstance();
conn.open('ext.db');
var store = new UserStore(conn);
```

B.8.3　创建表格

现在我们可以创建表格了，借助EXT提供的MVC模型，我们可以直接操作上节创建的store

对象，而无需对表格中已有的代码进行修改。

```
var selections = new Ext.grid.CheckboxSelectionModel();

var cm = new Ext.grid.ColumnModel([
    selections,
    {header:'编号',dataIndex:'userId',sortable:true},
    {header:'名称',dataIndex:'name',sortable:true},
    {header:'性别',dataIndex:'sex',sortable:true},
    {header:'备注',dataIndex:'descn',sortable:true}
]);

var grid = new Ext.grid.GridPanel({
    autoHeight: true,
    store: store,
    cm: cm,
    sm: selections,
    renderTo: 'grid',
    tbar: new Ext.Toolbar(['-', {
          text: '添加一行',
          handler: function(){
              var user = {
                  userId: new Date().getTime(),
                  name:'name',
                  sex:'sex',
                  descn:'descn'
              };
              store.addUser(user);
          }
    }, '-', {
        text: '删除一行',
        handler: function(){
            Ext.Msg.confirm('信息', '确定要删除？', function(btn){
                if (btn == 'yes') {
                    selections.each(function(s){
                        store.remove(s);
                    });
                }
            });
        }
    }, '-'])
});
store.load();
```

这些步骤与普通的EXT基本相同，先建立ColumModel和SelectionModel，然后拼合表格。bbar部分设置了两个按钮，一个用于添加用户数据，另一个用于删除选中行的数据。在执行添加和删除操作时，Proxy也会自动同步数据库中的数据。

到此为止，一个完整的表格已经完成。

示例在air/07_grid中。

附录 C

EXT的版本变迁

本章内容

- EXT开发计划
- EXT 2.1
- EXT 2.2
- EXT 2.2.1

C.1 EXT 开发计划

2008年3月20日，EXT官方网站公布了EXT在2008年至2009年的开发计划。直至2008年11月13日，已经有EXT 2.1和EXT 2.2两个版本相继发布。我们可以通过下面这份对EXT开发计划的翻译来了解EXT从2.*x*到3.*x*的升级过程。在这次升级过程中，主要改写了事件注册模型，并为后台服务器提供了更多的支持。

EXT 开发计划（2008~2009）

我们2008年的目标是继续完善EXT的2.*x*版本，加入新的组件并增强现有部分组件的功能。在明年即将发布的3.0版本中，我们将加入一些新的功能。除了经常被提及的图表、Comet等特性外，我们还将重点关注EXT组件与服务器集成。除了将插件和示例绑定到现有的服务器框架上，我们还将推出自己的综合传输机制，这样就可以在EXT组件与你选择的后台系统之间进行透明的数据绑定和数据编组。

注意，下面列出的计划可能会随时改变，这些并不是我们计划实现的具体功能，而是我们未来12个月内的开发计划。在这期间，也许还会根据需要产生额外的维护版本。

EXT 2.1（2008春）

- 完全支持REST
- 远程组件读取示例
- 为表单提供非Ajax形式的提交功能
- 将所有适配器库更新到当前版本
- 对AIR平台提供支持
- 表格过滤器（用于搜索）

- 复选框/单选框
- 多选控件
- 状态栏

EXT 2.2（2008夏）

- 加强Ext.Ajax
- 为EXT组件提供复位样式的功能
- 提供树形加载数据的更多实例（包括加载XML格式数据的实例）
- 支持浏览器历史记录
- 滑动条组件
- 文件上传控件

EXT 3.0（2008冬/2009初）

- 更新EXT事件注册模块
- Flash图表API
- 支持Comet/Bayeux
- 为EXT组件集成客户端服务器的数据绑定/编组
- 第三方RPC的演示示例（比如DWR）
- 提高可访问性

C.2　EXT 2.1

EXT 2.1的主要更新如下所示。

- 完全支持REST
- 状态栏
- 滑动条
- 远程读取组件配置
- 表格查询插件
- 布局浏览器
- Spotlight示例

C.2.1　完全支持 REST

REST（Representational State Transfer，表述性状态传输）是一种Web Service风格的架构模式，它可以完全调用HTTP语义。下面我们来看看EXT是如何为REST提供支持的。

1. 使用GET获得用户信息

```
Ext.Ajax.request({
    url: '/users',
    method: 'GET'
})
```

2. 使用POST添加一篇新文章

```
Ext.Ajax.request({
    url: '/articles',
    method: 'POST',
    params: {
        author: 'Patrick Donelan',
        subject: 'RESTful Web Services'
    }
})
```

3. 使用PUT更新文章信息

```
Ext.Ajax.request({
    url: '/articles/restful-web-services',
    method: 'PUT',
    params: {
        author: 'Patrick Donelan',
        subject: 'RESTful Web Services are easy with Ext!'
    }
})
```

4. 使用DELETE删除文章信息

```
Ext.Ajax.request({
    url: '/articles/rpc-is-the-best-web-architecture',
    method: 'DELETE',
    success: function(){ alert('Begone!'); }
})
```

为了避免缓存影响操作，可以利用下面的语句禁用缓存。

```
Ext.Ajax.disableCaching = false;
```

为了返回JSON数据，可以设置默认的首部信息。

```
Ext.Ajax.defaultHeaders = {
    'Accept': 'application/json'
};
```

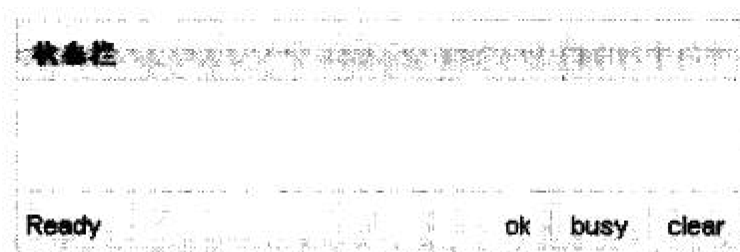
在访问受保护的资源时，需要在首部中添加授权信息，如下所示。

```
Ext.Ajax.defaultHeaders.Authorization = "Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==";
Ext.Ajax.request({
    url: '/protected_content',
    method: 'GET'
})
```

C.2.2 状态栏

实际上，状态栏（Statusbar）就是放在bbar上的工具条。顾名思义，我们可以在它上面执行许多与状态有关的操作。下面我们来看看图C-1中的效果。

状态栏的实现过程与工具条的实现过程一样，先创建一个Panel，然后把状态栏放到对应的bbar位置上，如下面的代码所示。



图C-1 状态栏控件的效果

```
new Ext.Panel({
    title: 'StatusBar',
    width: 300,
    height: 100,
    renderTo: 'status',
    bbar: new Ext.StatusBar({
        id: 'my-status',
        defaultText: 'Default status text',
        defaultIconCls: 'default-icon',
        text: 'Ready',
        iconCls: 'ready-icon',
        items: [{
            text: 'ok',
            handler: function() {
                Ext.getCmp('my-status').setStatus({
                    text: 'OK',
                    iconCls: 'ok-icon',
                    clear: true
                });
            }
        }, '-', {
            text: 'busy',
            handler: function() {
                Ext.getCmp('my-status').showBusy();
            }
        }, '-', {
            text: 'clear',
            handler: function() {
                Ext.getCmp('my-status').clearStatus();
            }
        }]
    })
});
```

状态栏比工具条多了几个参数，如下所示。

- text和iconCls参数，它们分别是状态栏初始化后显示的文字和图标。
- defaultText和defaultIconCls参数，表示默认显示的文字和图标。

初始化和默认的区别在于：状态栏第一次在页面上渲染时，首先显示的就是text和iconCls。在用setStatus()函数修改了状态栏的状态信息后，一段时间后会自动清空状态信息，这时显示的就是defaultText和defaultIconCls了。

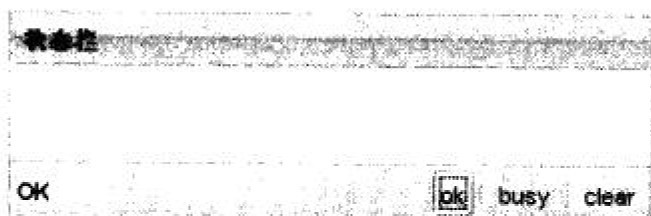
下面我们来看看在状态栏里设置的3个按钮，如下所示。

- ok按钮的处理函数会调用setStatus()函数将状态栏左边的文字改为“OK”，图标改为

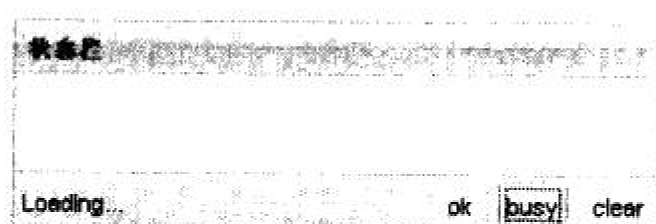
ok-icon。

参数clear表示这段状态文字和图标在一段时间后会自动消失，并显示默认的文字和图标。单击ok按钮后的效果如图C-2所示。

- busy按钮的处理函数会调用showBusy()函数，这将显示"Loading..."和相应的图标。因为这个功能很常用，所以StatusBar中内置了这个状态，我们可以通过showBusy()直接调用它，效果如图C-3所示。

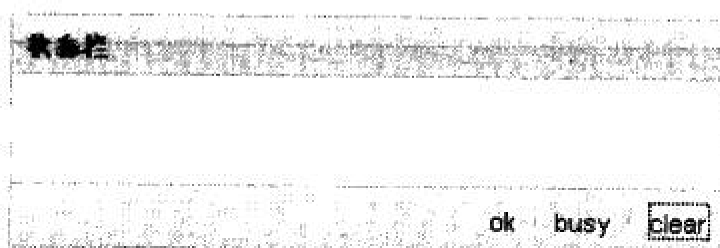


图C-2 单击ok按钮后显示的效果



图C-3 调用showBusy()函数后的显示效果

- clear按钮的处理函数会调用clearStatus()函数。它的功能很简单，直接清空状态文字和图标，如图C-4所示。



图C-4 调用clearStatus()函数清空状态文字和图标

有关状态栏的基本应用就讲到这里，我们使用的示例在2.1/1-statusbar.html中。建议大家去看看ext-2.1/examples/statusbar/目录下的示例，其中ValidationStatus的效果非常令人惊叹。

C.2.3 滑动条

滑动条（Slider）直接继承自Ext.BoxComponent，而不是继承自Ext.form.Field。虽然它也有value，可以使用setValue()赋值和getValue()取值，但是它不属于Ext.form.Field的继承体系。

创建滑动条十分简单，只需要知道它的宽度、最大值和最小值即可，具体方法如下所示：

```
new Ext.Slider({
    renderTo: 'slider1',
    width: 214,
    minValue: 0,
    maxValue: 100
});
```

这样我们就可以得到一个宽度为214（像素），最小值为0，最大值为100的滑动条了，如图C-5所示。

可以通过value设置滑动条的初始值，还可以使用increment来设定滑动的步长，如下所示：

```
new Ext.Slider({
    renderTo: 'slider2',
    width: 214,
    minValue: 0,
    maxValue: 100,
    value:50,
    increment: 10
});
```

这里设置的初始值为50，滑动块显示在正中间，如图C-6所示。



图C-5　滑动条显示效果



图C-6　为滑动条设置初始值

我们还可以实现竖直的滑动条，只需要我们修改两个参数：vertical: true和height: 214，如下所示。

```
new Ext.Slider({
    renderTo: 'slider3',
    height: 214,
    minValue: 0,
    maxValue: 100,
    vertical: true
});
```

既然滑动条是竖直的，就不能设置宽度了，而应该设置高度。我们设置滑动条的高度为214（像素），最小值0表示在底部，最大值100表示在顶部，如图C-7所示。

示例在2.1/2-slider.html中。

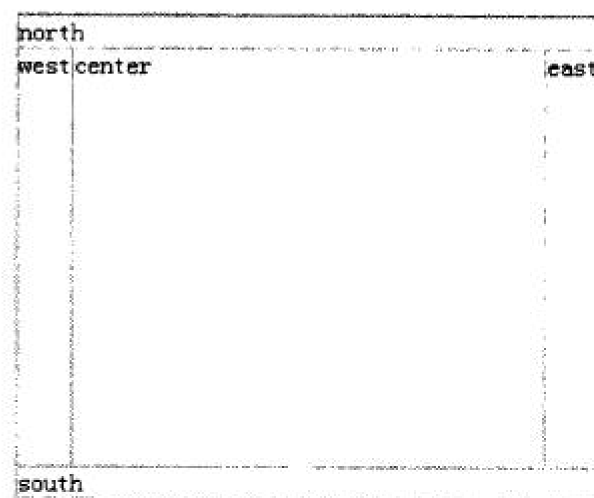
建议大家参考ext-2.1/examples/slider/目录下的示例，其中为`Slider`编写了扩展，可以实时显示拖动的数值，还有用CSS修改滑动条显示样式的示例。



图C-7　竖直的滑动条

C.2.4　远程读取组件配置

远程读取组件配置（ComponentLoader）并不是一个包含在ext-all.js中的组件，我们在示例中看到的是`Ext.ux.ComponentLoader`的扩展组件。通过它，我们可以使用从后台读取的JSON数据来对页面的组件进行初始化。实际上，在它的内部还是使用了`Ext.ComponentMgr`，可惜ext-2.1/examples/remoteload/目录下只有用PHP实现的示例。我们写了一个HTML的示例，通过`localXHR.js`读取本地文本文件中的JSON数据获得配置，然后为页面创建一个`layout:'border'`的Viewport组件，如图C-8所示。



图C-8　远程读取配置生成的Viewport

看一下2.1/3-remoteload.html中的JavaScript脚本，如下所示。

```
Ext.ux.ComponentLoader.load({
    url: '3-remoteload.txt',
    params: {
        testing: 'Testing params'
    }
});
```

上述代码演示了Ext.ux.ComponentLoader使用Ajax读取3-remoteload.txt中的数据，并传递testing:'Testing params'的参数。当然，我们这里用不到参数。

前台代码只有这些，具体的配置代码在3-remoteload.txt中，如下所示：

```
{components:[{
    xtype:'viewport',
    layout:'border',
    items:[{
        region:'north',
        html:'north'
    },{
        region:'south',
        html:'south'
    },{
        region:'west',
        html:'west'
    },{
        region:'east',
        html:'east'
    },{
        region:'center',
        html:'center'
    }]
}],success:true}
```

可以看出，这是一段完整的JSON数据，需要注意其中的两个部分：success:true表明返回成功，只有成功的情况下才会继续处理；components是具体的配置部分，我们用到的Viewport就包含在其中。

通过这种方式，我们可以让后台为前台提供需要的组件和布局信息，这为用户自定义界面提供了很大的帮助。这里只演示了一个最简单的示例，推荐大家看一看ext-2.1/examples/remoteload/下的PHP示例。

C.2.5　表格过滤组件

表格过滤组件（GridFilter）的规模比较庞大，我们首先要在2.1/4-gridfilter.html中引用9个JavaScript脚本，然后才能在页面中使用这个表格过滤组件，如下面的代码所示：

```
<script type="text/javascript" src="./grid-filtering/menu/EditableItem.js"></script>
<script type="text/javascript" src="./grid-filtering/menu/RangeMenu.js"></script>
<script type="text/javascript" src="./grid-filtering/grid/GridFilters.js"></script>
<script type="text/javascript" src="./grid-filtering/grid/filter/Filter.js"></script>
<script type="text/javascript" src="./grid-filtering/grid/filter/StringFilter.js">
```

```
</script>
<script type="text/javascript" src="./grid-filtering/grid/filter/DateFilter.js">
</script>
<script type="text/javascript" src="./grid-filtering/grid/filter/ListFilter.js">
</script>
<script type="text/javascript" src="./grid-filtering/grid/filter/NumericFilter.js">
</script>
<script type="text/javascript" src="./grid-filtering/grid/filter/BooleanFilter.js">
</script>
```

这个表格过滤组件的功能很强大，它可以搜索string、date、list、numeric和boolean这5种类型的数据。它已经被设计成插件的形式，使用的时候可以直接设置到表格和PagingToolbar中，不需要改动原有的表格代码，只需要定义一个GridFilters并放到对应的plugins属性中即可，如下面的代码所示：

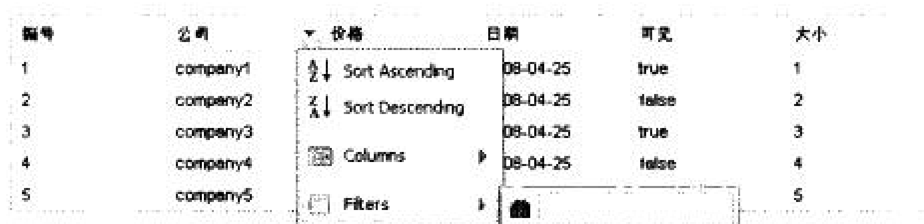
```
var filters = new Ext.grid.GridFilters({
  filters:[
    {type: 'numeric',  dataIndex: 'id'},
    {type: 'string',  dataIndex: 'company'},
    {type: 'numeric', dataIndex: 'price'},
    {type: 'date',  dataIndex: 'date'},
    {
        type: 'list',
        dataIndex: 'size',
        options: ['small', 'medium', 'large', 'extra large'],
        phpMode: true
    },
    {type: 'boolean', dataIndex: 'visible'}
]});
```

id和price字段使用的数字过滤器有大于、小于和等于3种情况，如图C-9所示。



图C-9　表格过滤组件——数字过滤器

Company字段使用的是字符串过滤器，可以输入文字进行模糊搜索，如图C-10所示。



图C-10　表格过滤组件——字符串过滤器

Date字段使用的是日期过滤器，有3个时间段可供选择，如图C-11所示。



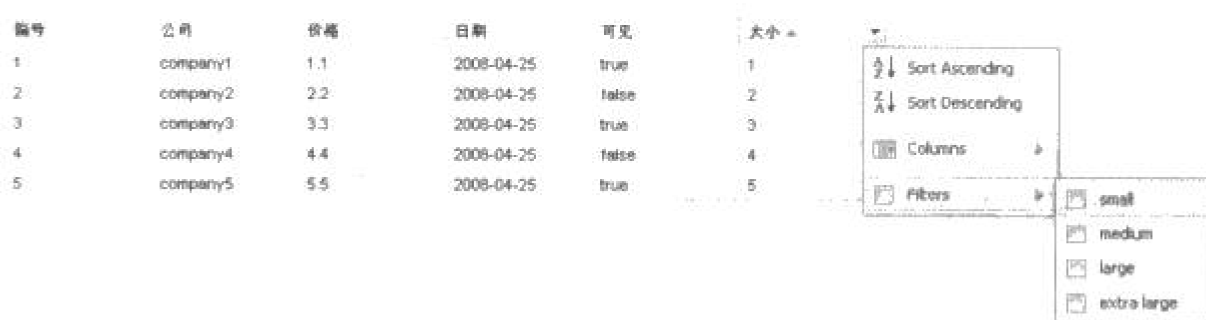
图C-11　表格过滤组件——日期过滤器

Size字段使用的是列表过滤器，自定义了几种搜索条件，如图C-12所示。



图C-12　表格过滤组件——列表过滤器

visible使用的是布尔过滤器，通过radioMenu选择true或false，如图C-13所示。



图C-13　表格过滤组件——布尔过滤器

有一个问题需要注意，因为是扩展的原因，默认的resources下没有包含需要的图片，我们只能在JavaScript代码中手工指定了，如下面的代码所示：

```
Ext.menu.RangeMenu.prototype.icons = {
    gt: './grid-filtering/img/greater_then.png',
    lt: './grid-filtering/img/less_then.png',
    eq: './grid-filtering/img/equals.png'
};
Ext.grid.filter.StringFilter.prototype.icon = './grid-filtering/img/find.png';
```

C.2.6　布局浏览器

通过ext-2.1/examples/layout-browser目录下的新示例可以浏览所有的布局方式，如下所示。

- 基本布局方式
  基本布局方式有Absolute、Accordion、Anchor、Border、Card(TabPanel)、Card(Wizard)、Column Fit、Form和Table等9种。
- 自定义布局方式
  自定义布局方式有Row、Center两种。
- 综合示例
  综合示例有Absolute Layout Form和Tabs with Nested Layouts两个。

它的参考价值远大于它演示的那几种布局，建议大家认真研究一下这个示例。

C.2.7　Spotlight 示例

又是一个绚丽的示例，使用ext-2.1/examples/core/下的Spotlight，用动画效果将整个背景遮罩起来，每次只显示当前允许编辑的Panel。显示完每个Panel之后，又以动画的效果撤销遮罩效果。如果将该示例用在提示信息的场合，也许又会要让许多用户感到惊讶。它在ext-2.1/examples/core/目录下，建议认真看一下。

至此为止，EXT 2.1中的所有更新都已介绍完了，接下来我们看看EXT 2.2中的更新。

C.3　EXT 2.2

EXT 2.2的主要更新如下所示。

- 完全支持Firefox 3.0
- 添加了Ext.History组件和示例
- 完全重构复选框和单选框
- 添加了CheckboxGroup和RadioGroup组件以及示例
- 添加了多选框和ItemSelector扩展以及示例
- 添加了文件上传组件和示例
- 添加了XmlTreeLoader扩展和示例
- 添加了一些新的拖放示例
- 添加了GMapPanel扩展和示例
- 提升了表格的性能
- 补充文档内容
- 更新不同的国际化文件

C.3.1　添加 **Ext.History** 组件和示例

Ext.History组件提供了记忆浏览历史的功能，在Ajax中本来无法使用浏览器的“后退”和“前进”按钮。有了Ext.History后，我们可以把页面上的组件注册到Ext.History里，使用浏

览器的“后退”和“前进”按钮控制历史操作。

Ext.History的使用步骤如下所示。

(1) 做准备工作

首先在HTML中加上input和iframe标签，供Ext.History使用。

```
<form id="history-form" class="x-hidden">
    <input type="hidden" id="x-history-field" />
    <iframe id="x-history-frame"></iframe>
</form>
```

然后在JavaScript中对Ext.History进行初始化，如下所示。

```
Ext.History.init();
```

因为input和iframe的标签都使用了默认值，所以初始化时不需要设置fieldId和iframeId。

(2) 创建一个Ext.Window，监听移动事件，把移动事件加入到Ext.History中作为记录，如下所示。

```
var win = new Ext.Window({
    title: 'window',
    width: 300,
    height: 100
});

win.on('move', function(component, x, y) {
    Ext.History.add(x + "," + y);
});
```

在监听事件的函数中，每当win发生移动时，就把x坐标和y坐标保存到Ext.History中。

监听Ext.History的change事件，在浏览器后退或前进时修改win的位置，如下所示。

```
Ext.History.on('change', function(token) {
    if (token) {
        var position = token.split(",");
        win.setPagePosition(position[0], position[1]);
    }
});
```

token就是保存到Ext.History中的历史记录，我们保存的格式是"x,y"的字符串，处理时先把token切分成两段，获得对应的x坐标和y坐标，然后再调用setPagePosition()函数修改win的位置。

经过上述步骤后，我们就可以在拖动窗口之后使用浏览器的“后退”和“前进”按钮控制窗口的历史了。

示例在2.2/1-history.html中。

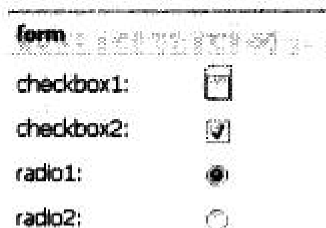
C.3.2 完全重构复选框和单选框

EXT 2.2里重写了Ext.form.Checkbox和Ext.form.Radio，现在这两个组件变得非常漂亮

了如图C-14所示。

幸运的是，需要修改的部分只涉及显示效果，继承结构和主要API都没有发生改变。Ext.form.Checkbox中专门添加了一系列用来控制显示效果的属性，可以通过自定义CSS来控制复选框和单选框的显示方式。

checkedCls、focusCls、mouseDownCls和overCls分别用来指定对应状态下的CSS样式。



图C-14　新版本的复选框和单选框

下面的是生成复选框和单选框的代码：

```
var form = new Ext.form.FormPanel({
    title: 'form',
    frame: true,
    items: [{
        xtype: 'checkbox',
        name: 'checkbox1',
        fieldLabel: 'checkbox1'
    }, {
        xtype: 'checkbox',
        name: 'checkbox2',
        fieldLabel: 'checkbox2'
    }, {
        xtype: 'radio',
        name: 'radio',
        fieldLabel: 'radio1'
    }, {
        xtype: 'radio',
        name: 'radio',
        fieldLabel: 'radio2'
    }]
});
```

C.3.3　添加 CheckboxGroup 和 RadioGroup 组件及其示例

CheckboxGroup和RadioGroup的出现，给广大开发者带来了惊喜，现在我们可以为复选框和单选框实现各种复杂的排列方式了，如下所示。

1. 默认排列方式（横排）

以前无法直接让复选框和单选框横向排列，即便是使用columnLayout，实现过程也很复杂。现在有了CheckboxGroup和RadioGroup，默认的排序方式就是横排了，如图C-15所示。



图C-15　CheckboxGroup的默认排序方式

实现过程如下面的代码所示：

```
{
    xtype: 'checkboxgroup',
```

```
        fieldLabel: '自动布局',
        items: [
            {boxLabel: 'Item 1', name: 'cb-auto-1'},
            {boxLabel: 'Item 2', name: 'cb-auto-2', checked: true},
            {boxLabel: 'Item 3', name: 'cb-auto-3'},
            {boxLabel: 'Item 4', name: 'cb-auto-4'},
            {boxLabel: 'Item 5', name: 'cb-auto-5'}
        ]
}
```

2. 竖直排列方式

如果还是希望使用竖排的形式，则需要设置column属性，效果如图C-16所示。



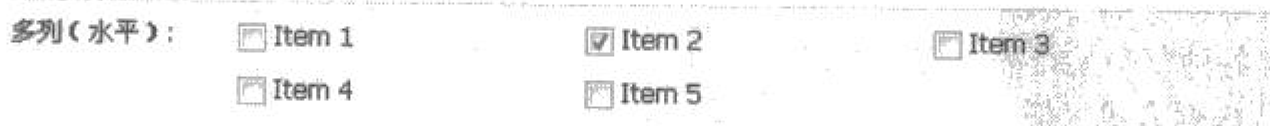
图C-16 CheckboxGroup的竖排方式

实现过程如下面的代码所示：

```
{
            xtype: 'checkboxgroup',
            fieldLabel: '单列',
            itemCls: 'x-check-group-alt',
            columns: 1,
            items: [
                {boxLabel: 'Item 1', name: 'cb-col-1'},
                {boxLabel: 'Item 2', name: 'cb-col-2', checked: true},
                {boxLabel: 'Item 3', name: 'cb-col-3'}
            ]
        }
```

3. 多列排列方式

既然可以指定单列，也就可以指定多列，如图C-17所示。



图C-17 CheckboxGroup多列排列方式

实现过程如下面的代码所示：

```
{
            xtype: 'checkboxgroup',
            fieldLabel: '多列（水平）',
            columns: 3,
            items: [
                {boxLabel: 'Item 1', name: 'cb-horiz-1'},
                {boxLabel: 'Item 2', name: 'cb-horiz-2', checked: true},
```

```
                {boxLabel: 'Item 3', name: 'cb-horiz-3'},
                {boxLabel: 'Item 4', name: 'cb-horiz-4'},
                {boxLabel: 'Item 5', name: 'cb-horiz-5'}
            ]
    }
```

上例中，子元素是按行水平排列的，也可以设置成按列垂直排列，如图C-18所示。



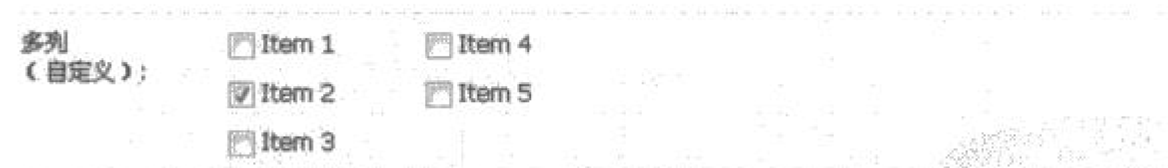
图C-18　CheckboxGroup列优先排列方式

实现过程如下面的代码所示：

```
{
        xtype: 'checkboxgroup',
        fieldLabel: '多列（竖直）',
        itemCls: 'x-check-group-alt',
        columns: 3,
        vertical: true,
        items: [
            {boxLabel: 'Item 1', name: 'cb-vert-1'},
            {boxLabel: 'Item 2', name: 'cb-vert-2', checked: true},
            {boxLabel: 'Item 3', name: 'cb-vert-3'},
            {boxLabel: 'Item 4', name: 'cb-vert-4'},
            {boxLabel: 'Item 5', name: 'cb-vert-5'}
        ]
    }
```

4. 自定义多列排列方式

也可以根据需要自定义多列，效果如图C-19所示。



图C-19　CheckboxGroup自定义多列排列方式

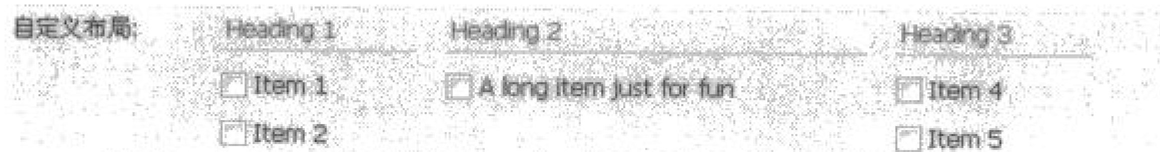
```
{
    xtype: 'checkboxgroup',
    fieldLabel: '多列<br />（自定义）',
    columns: [100, 100],
    vertical: true,
    items: [
        {boxLabel: 'Item 1', name: 'cb-custwidth', inputValue: 1},
        {boxLabel: 'Item 2', name: 'cb-custwidth', inputValue: 2, checked:
         true},
        {boxLabel: 'Item 3', name: 'cb-custwidth', inputValue: 3},
        {boxLabel: 'Item 4', name: 'cb-custwidth', inputValue: 4},
        {boxLabel: 'Item 5', name: 'cb-custwidth', inputValue: 5}
```

```
    ]
}
```

columns部分定义的是高度和宽度，也可以使用百分比来表示。

5. 自定义布局排列方式

还可以使用自定义布局排列方式，如图C-20所示。



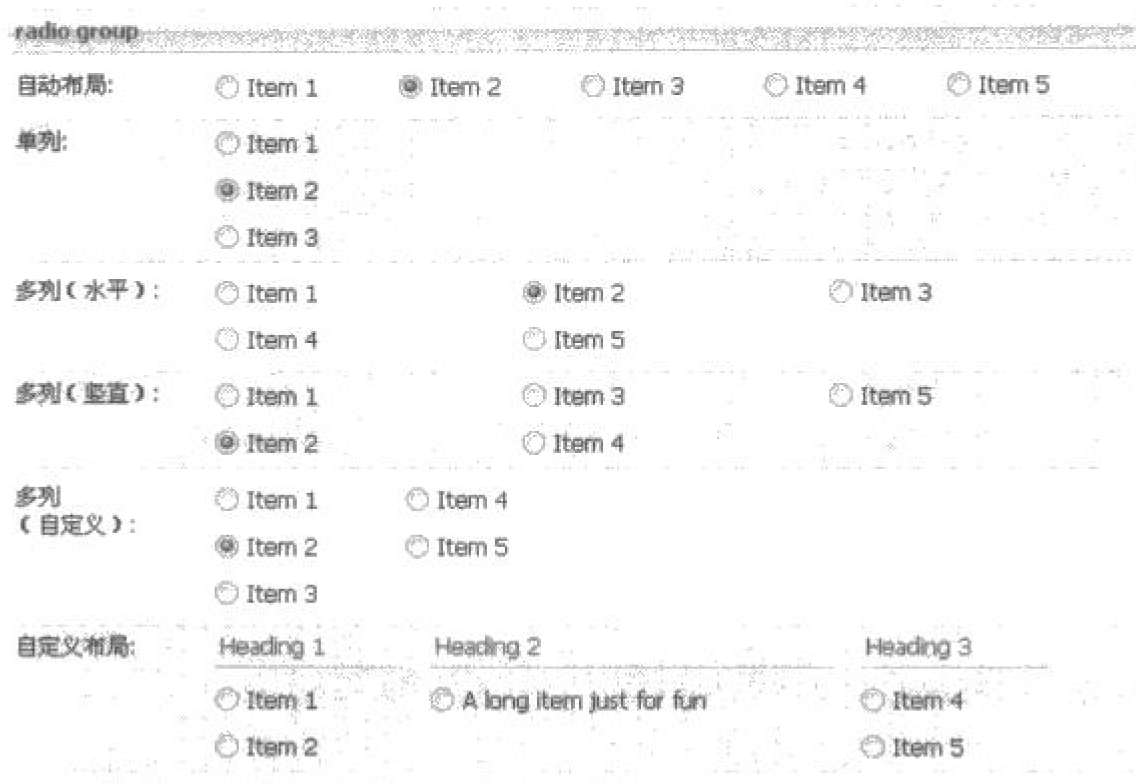
图C-20 CheckboxGroup自定义布局

实现过程如下面的代码所示：

```
{
        xtype: 'checkboxgroup',
        itemCls: 'x-check-group-alt',
        fieldLabel: '自自定义布局',
        allowBlank: false,
        anchor: '95%',
        items: [{
            columnWidth: '.25',
            items: [
                {xtype: 'label', text: 'Heading 1', cls:'x-form-check-group-
                label', anchor:'-15'},
                {boxLabel: 'Item 1', name: 'cb-cust-1'},
                {boxLabel: 'Item 2', name: 'cb-cust-2'}
            ]
        },{
            columnWidth: '.5',
            items: [
                {xtype: 'label', text: 'Heading 2', cls:'x-form-check-group-
                label', anchor:'-15'},
                {boxLabel: 'A long item just for fun', name: 'cb-cust-3'}
            ]
        },{
            columnWidth: '.25',
            items: [
                {xtype: 'label', text: 'Heading 3', cls:'x-form-check-group-
                label', anchor:'-15'},
                {boxLabel: 'Item 4', name: 'cb-cust-4'},
                {boxLabel: 'Item 5', name: 'cb-cust-5'}
            ]
        }
```

还可以在组中使用子布局，从而实现更复杂的布局。

RadioGroup与CheckboxGroup的情况完全相同，图C-21是RadioGroup的显示效果。



图C-21　RadioGroup的各种布局效果

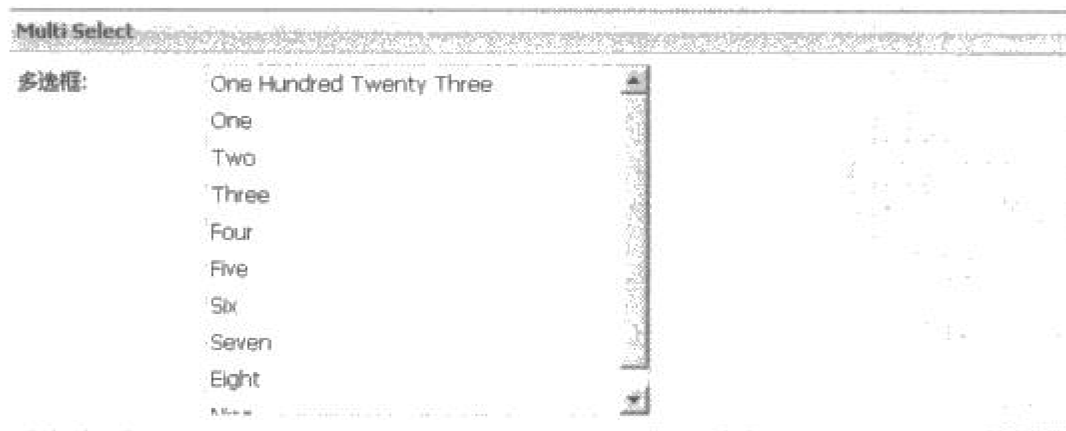
C.3.4　添加多选框和 ItemSelector 扩展以及示例

Ext 2.2中的多选框（MultiSelect）和ItemSelector都是扩展控件，需要在HTML中引入对应的MultiSelect.js和ItemSelector.js，然后才能使用。

多选框是为了补充Combo中没有实现的多选功能，效果如图C-22所示。

为了使用多选框，我们要先为HTML引入对应的CSS和JavaScript文件，如下所示。

```
<link rel="stylesheet" type="text/css" href="../../ux/css/multiselect.css"/>
<script type="text/javascript" src="../../ux/css/MultiSelect.js"></script>
```



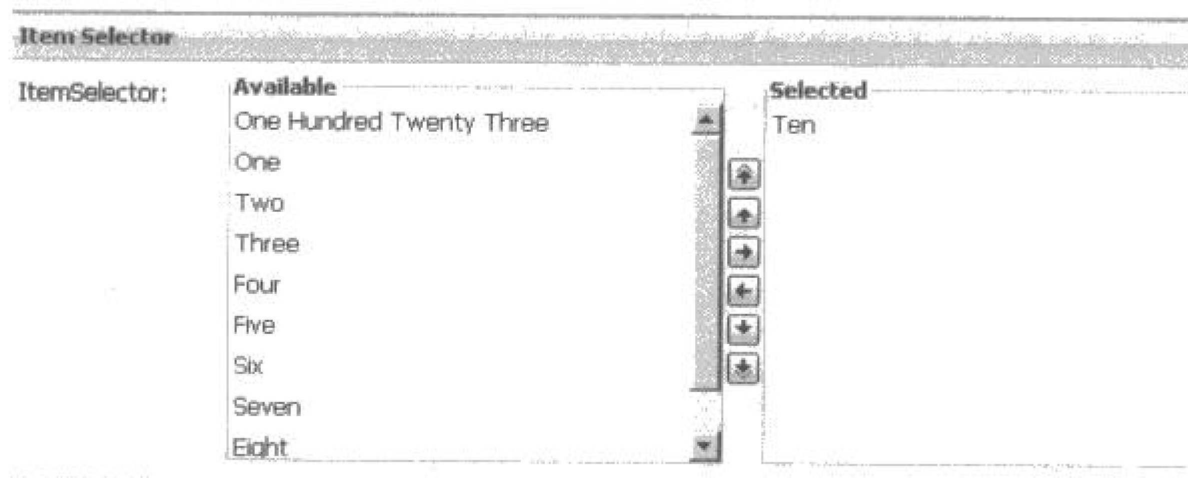
图C-22　多选控件的显示效果

然后使用与Combo相似的语法创建多选框，如下所示：

```
var form = new Ext.form.FormPanel({
    title: 'Multi Select',
```

```
        frame: true,
        width: 600,
        items: [{
            xtype:"multiselect",
            fieldLabel:"多选框",
            name:"multiselect",
            dataFields:["code", "desc"],
            valueField:"code",
            displayField:"desc",
            width:250,
            height:200,
            allowBlank:false,
            data:[[123,"One Hundred Twenty Three"],
                ["1", "One"],["2", "Two"],["3", "Three"],["4", "Four"],["5", "Five"],
                ["6", "Six"], ["7", "Seven"], ["8", "Eight"], ["9", "Nine"]]
        }]
});
```

ItemSelector也是很常用的一种多选排列组件，效果如图C-23所示。



图C-23　ItemSelector的显示效果

它支持左右移动、上下移动、全选、清空、移动到顶部和底部等操作。因为在它的内部使用了多选框，所以要将多选框需要的外部CSS和JavaScript都添加到HTML中，如下面的代码所示：

```
<link rel="stylesheet" type="text/css" href="../../ux/css/multiselect.css"/>
<script type="text/javascript" src="../../ux/css/MultiSelect.js"></script>
<script type="text/javascript" src="../../ux/css/ItemSelector.js"></script>
```

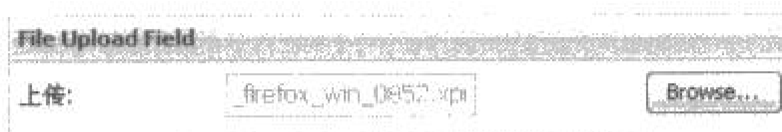
示例代码如下所示：

```
var form = new Ext.form.FormPanel({
    title: 'Item Selector',
    frame: true,
    width: 600,
    items: [{
        xtype:"itemselector",
        name:"Item Selector",
        fieldLabel:"ItemSelector",
        dataFields:["code", "desc"],
        toData:[["10", "Ten"]],
```

```
            msWidth:250,
            msHeight:200,
            valueField:"code",
            displayField:"desc",
            imagePath:" ../../ux/images/",
            toLegend:"Selected",
            fromLegend:"Available",
            fromData:[[123,"One Hundred Twenty Three"],
                ["1","One"],["2","Two"],["3", "Three"],["4","Four"],["5","Five"],
                ["6", "Six"], ["7", "Seven"], ["8", "Eight"], ["9", "Nine"]],
        }]
    });
```

C.3.5 添加文件上传组件和示例

FileUploadField是美化了的文件上传选择控件，它让之前简陋的input type="file"更接近EXT的整体风格，效果如图C-24所示。



图C-24 文件上传控件的显示效果

为了使用它，我们还是要加入额外的CSS和JavaScript文件。

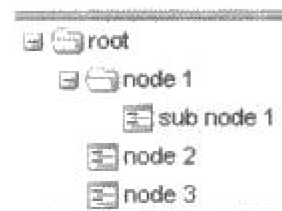
```
<link rel="stylesheet" type="text/css" href="../../ ux/fileuploadfield/css/fileuploadfield.css"/>
<script type="text/javascript" src="../../ux/fileuploadfield/FileUploadField.js"></script>
```

注意，使用FileUploadField时，不要忘记为表单设置fileUpload:true，如下面的代码所示：

```
var form = new Ext.form.FormPanel({
    title: 'File Upload Field',
    fileUpload: true,
    frame: true,
    width: 400,
    items: [{
        xtype:"fileuploadfield",
        fieldLabel:"上传",
        name:"fileuploadfield"
    }]
});
```

C.3.6 添加 XmlTreeLoader 扩展和示例

以前TreeLoader异步加载节点信息只支持JSON格式的数据，现在用XmlTreeLoader也可以读取XML格式的数据了。读取XML数据的XmlTreeLoader的最终显示效果如图C-25所示。



图C-25 读取XML数据的XmlTreeLoader最终显示效果

首先在HTML中添加XmlTreeLoader.js的引用，如下所示：

```
<script type="text/javascript" src="../../ux/XmlTreeLoader.js"></script>
```

然后制作一个为树提供数据的XML文件，如下所示：

```
<root>
    <node text="node 1">
        <sub text="sub node 1" leaf="true"/>
    </node>
    <node text="node 2" leaf="true"/>
    <node text="node 3" leaf="true"/>
</root>
```

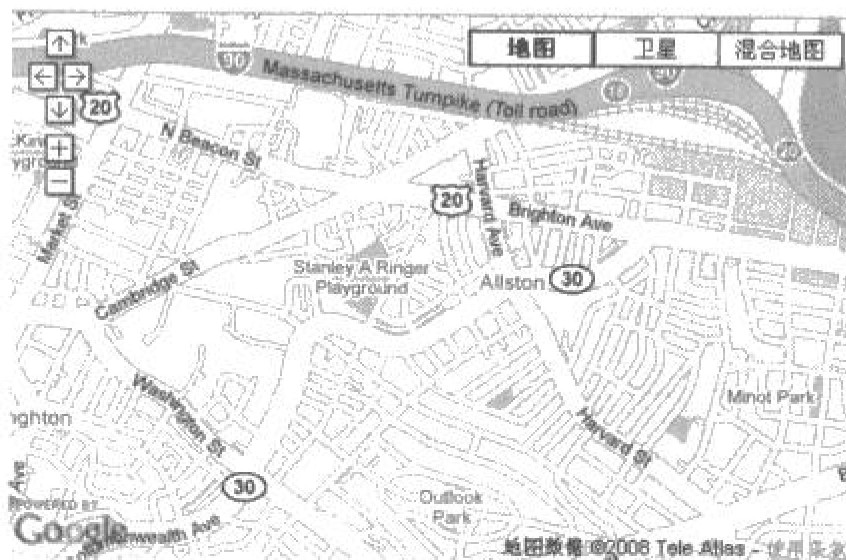
最后在JavaScript中使用XmlTreeLoader读取数据，如下所示：

```
var tree = new Ext.tree.TreePanel({
    autoHeight: true,
    loader: new Ext.ux.XmlTreeLoader({
        dataUrl: '6-xmltreeloader.xml',
        processAttributes : function(attr){
            if (attr.tagName == 'node') {
                attr.loaded = true;
            }
        }
    }),
    root: new Ext.tree.AsyncTreeNode({
        text: 'root'
    })
});
```

dataUrl表示从'6-xmltreeloader.xml'获取数据，processAttributes用于在读取数据之后，先由用户处理，然后再创建树节点，我们为所有tagName="node"的节点设置了loaded: true属性，这些XML中的节点会对应生成叶子节点，避免出现循环加载。

C.3.7　添加 GMapPanel 扩展和示例

EXT 2.2里还提供了一个调用Google Map的示例，如图C-26所示。



图C-26　与Google Map整合的显示效果

首先要把Google Map的引用链接和GMapPanel.js加入HTML中，如下所示：

```
<script src="http://maps.google.com/maps?file=api&amp;v=2.x&amp;key=
ABQIAAAAJDLv3q8BFBryRorw-851MRT2yXp_ZAY8_ufC3CFXhHIE1NvwkxTyuslsNlFqyphYqv
1PCUD8WrZA2A" type="text/javascript"></script>
<script src="../../ux/Ext.ux.GMapPanel.js"></script>
```

下面是Ext.ux.GMapPanel的引用，有了这些就可以使用Ext.ux.GMapPanel了，如下所示：

```
var panel = new Ext.ux.GMapPanel({
    zoomLevel: 14,
    gmapType: 'map',
    mapConfOpts: ['enableScrollWheelZoom','enableDoubleClickZoom',
    'enableDragging'],
    mapControls: ['GSmallMapControl','GMapTypeControl','NonExistantControl'],
    setCenter: {
        geoCodeAddr: '4 Yawkey Way, Boston, MA, 02215-3409, USA',
        marker: {title: 'Fenway Park'}
    },
    markers: [{
        lat: 42.339641,
        lng: -71.094224,
        marker: {title: 'Boston Museum of Fine Arts'},
        listeners: {
            click: function(e){
                Ext.Msg.alert('Its fine', 'and its art.');
            }
        }
    },{
        lat: 42.339419,
        lng: -71.09077,
        marker: {title: 'Northeastern University'}
    }]
});
```

C.4　EXT 2.2.1

EXT 2.2.1的主要更新如下所示。

- 提供对Chrome的支持
- 解决了一些内存泄露问题
- 为Ext.Container添加了removeAll()
- 为air提供了很多新的组件

C.4.1　提供对 Chrome 的支持

从官方提供的更新列表来看，只有在Ext.js中添加了Ext.isIE8和Ext.isChrome，这就可以直接使用这两个属性判断当前使用的浏览器是否为IE8或Chrome了。

在页面进行初始化时，Ext.EventManager.js为BODY标签设置名为ext-chrome的CSS样式。

再看ext-base.js和ext-all.js中的代码，就只有ext-base.js中包含了对Ext.isChrome的判断，也许是因为Chrome和FireFox太相似了么？反正结论是这条更新实在是没多少内容。

C.4.2 解决了一些内存泄露问题

EXT会导致内存泄露，页面中使用了EXT，浏览器的内存就会越占越多。这个问题导致目前不少项目不敢使用one page on application的方式，而是在页面中嵌套上IFRAME，用这种方式避免同一页面产生过多JavaScript对象的问题。

在此之前，有人提出过EXT中的内存泄露是因为事件模型设计上的问题造成监听函数占用的内存无法释放，也有人提出了解决内存泄露的方案。现在EXT官方终于放话说解决内存泄露了。

我们使用Ctrl + F在更新列表上只能找到两个与内存相关的项目。

(1) [Ext.form.TextField] - [Fixed minor memory leak when using autoSize.]

比较EXT 2.2和EXT 2.2.1中source/widgets/form目录下的TextField.js代码。发现在autoSize()这个函数的第397行多了一句Ext.removeNode(d)。这里被删除的变量d是在autoSize()函数中创建的一个DIV节点。它的用处是在修改TextField大小的时候，先将TextField的值作为一个文本节点放到DIV中，然后再获得DIV中的 innerHTML，以此来实现对转义字符的转换。而在EXT 2.2中，在d使用完之后，仅仅使用了d = null;这样的方式来告诉浏览器需要进行内存回收了。现在EXT 2.2.1中添加了Ext.removeNode(d)应该是显式地告诉浏览器把这个节点删除掉。以此来解决内存泄露的问题。

再去看一下source/core/Ext.js中的 removeNode()函数，发现原来在这里IE和其他浏览器是被区别处理的，如果是在非IE浏览器下，如果这个节点有父节点，就调用父节点的removeChild()函数删除该节点。我们上面讨论到的DIV节点是新创建出来的，还没有放到页面中，所以在非IE浏览器下，新增这段removeNode()是不会起任何作用的，这样我们的焦点就集中在了IE浏览器下。

在IE浏览器下，Ext.removeNode()函数的内容如下：

```
function(){
    var d;
    return function(n){
        if(n && n.tagName != 'BODY'){
            d = d || document.createElement('div');
            d.appendChild(n);
            d.innerHTML = '';
        }
    }
}();
```

它这里先在外部定义了一个临时变量d，这样所有的节点删除操作都可以共用这一个DIV标签了，然后嵌套的函数内部利用延迟加载，如果d还没初始化就创建一个DIV标签来用。

接下来就是IE中删除节点的方法了，我们要先把需要删除的节点放到这个共用的DIV标签中，然后调用d.innerHTML = '';进行删除。

感觉这样做似乎多此一举，可如果你把下面这段代码放到IE下多执行几遍，就会发现内存以

每次几十KB的速度慢慢增长。而且在函数的最后添加上removeNode()就可以解决IE浏览器下的内存泄露。

```
<script type="text/javascript">
removeNode = function(){
    var d;
    return function(n){
        if(n && n.tagName != 'BODY'){
            d = d || document.createElement('div');
            d.appendChild(n);
            d.innerHTML = '';
        }
    }
}();

function test() {
    var v = '<>';
    var d = document.createElement('div');
    d.appendChild(document.createTextNode(v));
    v = d.innerHTML;
    //removeNode(d); // 测试解决内存泄露的情况，可以去掉这行的注释
    d = null;
    //alert(v);
}
</script>

<button onclick="test()">button</button>
```

(2) [Ext.tree.AsyncTreeNode] - [Fixed a memory leak when reloading a node.]

比较EXT 2.2和EXT 2.2.1中source/widgets/tree目录下的AsyncTreeNode.js代码，发现只有104行存在改动。这行代码在EXT 2.2中是this.removeChild(this.firstChild)，在EXT 2.2.1中改成了 this.removeChild(this.firstChild).destroy()。在语句最后增添了一个destory()函数，意思应该是将刚刚删除的第一个节点销毁，释放内存。

由于Ext.tree.AsyncTreeNode继承自Ext.tree.TreeNode，而Ext.tree.TreeNode又继承自Ext.data.Node，所以这个this.removeChild()函数的具体实现还要去Ext.data.Node里去找。翻开Ext.data.Node的源代码，看到的只是在数据结构上删除了对应的节点，并重新计算第一个节点和最后一个节点的引用，里面没有删除节点释放内存的功能，这时我们突然想到树形中所有的节点在渲染后都由一个ui组件与内存中的数据模型相对应，实际中删除节点的功能应该是在ui组件中实现的。

Ext.tree.AsyncTreeNode 并没有定义自己的destroy()函数，它继承了Ext.tree.TreeNode中定义的destory()函数，在Ext.tree.TreeNode的destory()函数中首先调用所有子节点的destory()函数，然后判断是否存在 this.ui.destory()这个函数，如果当前节点对应的ui支持destory()函数，就调用destory()函数销毁ui释放内存。

实际销毁ui的destory()函数可以在Ext.tree.TreeNodeUI中找到。

```
destroy : function(){
    if(this.elNode){
```

```
            Ext.dd.Registry.unregister(this.elNode.id);
        }
        delete this.elNode;
        delete this.ctNode;
        delete this.indentNode;
        delete this.ecNode;
        delete this.iconNode;
        delete this.checkbox;
        delete this.anchor;
        delete this.textNode;

        if (this.holder){
             delete this.wrap;
             Ext.removeNode(this.holder);
             delete this.holder;
        }else{
            Ext.removeNode(this.wrap);
            delete this.wrap;
        }
    }
```

如上述代码所示，首先从拖曳注册中删除节点的注册信息，然后依次删除elNode、ctNode、indentNode、ecNode、iconNode、checkbox、anchor、textNode这些节点，最后删除this.holder和this.wrap。

这样来看，EXT之前的做法也太大意了，竟然在reload()的时候没有删除这些节点，不出内存泄露才叫奇怪了呢。想再现这个问题，只要创建一个TreePanel，然后不断执行rootNode.reload()就可以看到内存飙升了。

C.4.3 为 Ext.Container 添加了 removeAll()函数

Ext.Container里多了一个removeAll()函数，现在我们可以一次性删除容器里所有的组件了。它还可以使用一个autoDestroy参数，如果传入一个true，就会在删除容器内组件的同时把这些组件销毁掉。

从代码上来看，这只是添加了一个不怎么复杂的函数而已，但是因为太多组件都是继承自Ext.Container，为Ext.Container添加函数，无形中等于为所有继承自Ext.Container的组件都增加了这个可以清空内部组件的功能，所以EXT开发团队也把它单独列为一项重要的更新。

最容易演示的实例就是使用Ext.Panel，我们先在Ext.Panel里设置4个Panel，然后调用removeAll()函数把这4个Panel一次删除掉，如以下代码所示：

```
var panel = new Ext.Panel({
    items: [{
        html: '1'
    }, {
        html: '2'
    }, {
        html: '3'
    }, {
        html: '4'
    }]
```

```
});
panel.render('panel');

Ext.get('button').on('click', function() {
    panel.removeAll();
});
```

C.4.4　为 AIR 提供了很多新组件

1. Ext.sql.SQLLiteStore

支持操作数据库的简易Store。

在此之前我们想要从本地数据库获得数据，就只能自己定义一个Store类型，现在EXT为我们提供了一个封装好的类，只要我们把必须的参数传递进去，它就会自动帮我们完成数据库的连接配置。如果数据表不存在，还会帮咱们自动建立好表关系。使用起来也十分简单。

```
var store = new Ext.sql.SQLiteStore({
    dbFile: 'ext.db',
    tableName: 'user',
    key: 'userId',
    fields: User
});
```

需要传递的参数有本地数据库文件dbFile、数据表名tableName、表中的主键key和表中字段的定义fields。这些参数都是缺一不可的，偷懒省掉任何一个参数都会导致异常。

这里的dbFile参数中指定的文件并不一定要预先创建好，如果文件不存在，AIR会自动为我们生成一个文件的。剩下的就可以像从前一样操作我们的store了。我们的例子中演示的是如何在表格中使用这个SQLiteStore。

2. Ext.air.App

提供了一个便捷的途径来创建一个应用。Ext.air.App是一个单例，不需要使用new创建创建实例就可以直接使用，它为我们提供了两个函数。

- launchOnStartup(true) 会 调 用 air.NativeApplication.nativeApplication.startAtLogin=true，在用户登录的时候启动应用程序。（注，可能是在AIR安装到系统之后，可以设置自动启动的选项吧。）
- getActiveWindow()将返回当前活动的air窗口。但是使用AIR 1.5.1(winxp)测试的时候一直说"Error #2014: Feature is not available at this time."，查了一下官方文档，意思是这项功能在当前系统上还不支持。

```
Ext.air.App.launchOnStartup(true);
alert(Ext.air.App.getActiveWindow());
```

3. Ext.air.Clipboard

可以使用操作系统的剪贴板中的信息，也可以把特定格式的信息放到系统剪贴板中。

```
Ext.air.Clipboard.setData('air:text', 'text data');
var data = Ext.air.Clipboard.getData('air:text', 'originalPreferred');
alert(data);
```

使用setData()函数向剪贴板中复制数据的时候要先指定数据的格式，AIR现在支持的数据格式如下所示。

- air.ClipboardFormats.TEXT_FORMAT - 'air:text'
- air.ClipboardFormats.HTML_FORMAT - 'air:html'
- air.ClipboardFormats.RICH_TEXT_FORMAT - 'air:rtf'
- air.ClipboardFormats.URL_FORMAT - 'air:url'
- air.ClipboardFormats.FILE_LIST_FORMAT - 'air:file list'
- air.ClipboardFormats.BITMAP_FORMAT - 'air:bitmap'

在使用getData()函数从剪贴板中获取数据的时候，不但要指明数据类型，还需要设置数据传输模式。AIR现在支持的传输格式如下所示。

- air.ClipboardTransferMode.CLONE_ONLY - 'cloneOnly'，只返回源数据的副本。
- air.ClipboardTransferMode.CLONE_PREFERRED - 'clonePreferred'，如果剪贴板可以返回源数据的副本，就返回副本，否则返回源数据的引用。
- air.ClipboardTransferMode.ORIGINAL_ONLY - 'originalOnly'，只返回源数据的引用。
- air.ClipboardTransferMode.ORIGINAL_ONLY - 'originalPreferred'，如果剪贴板可以返回源数据的引用，就返回引用，否则返回源数据的副本。

这里还是遇到了问题，使用setData()函数的确可以把数据保存到系统剪贴板中，在记事本中使用Ctrl + V可以看到刚刚放到剪贴板中的数据，但是使用getData()却无论如何也得不到剪贴板中的内容，总是显示'undefined'。

4. Ext.air.Debug

这是一个十分方便的提示函数，可以把调试数据显示到AIR的控制台中。它支持两个参数，第一个参数是需要输出的对象，第二个参数是调试信息前面留出的缩进数，每个缩进都是一个制表符（tab）的距离。实例代码如下：

```
Ext.air.dir('click', 1);
Ext.air.dir({
    name: 'name',
    age: 18
}, 2);
```

在使用中发现了一个问题，那就是EXT总是把第一个参数当做对象，即使第一个参数的类型是字符串，它也会把它分拆成好几部分进行显示。因此Ext.air.dir('click', 1);就显示成了下面这样：

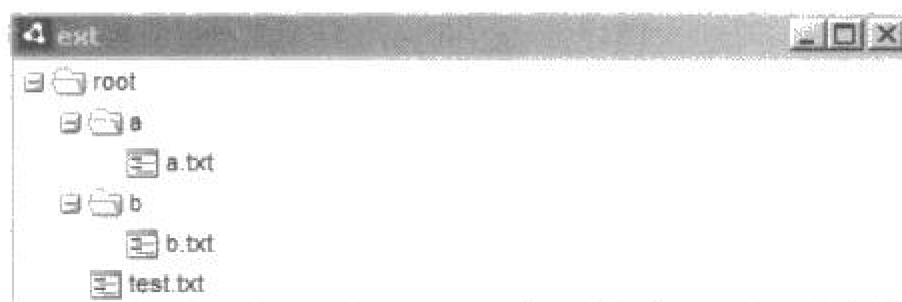
```
0: c
1: l
2: i
3: c
4: k
toggle: function (b, a) {return this==b?a:b}
trim: function () {return this.replace(a,"")}
```

而在实际对象中包含的字符串参数却可以正常显示出来。看来EXT中还要加一个类型判断才好啊。`Ext.air.dir({name: 'name', age: 18}, 2);`的显示结果如下：

```
name: name
age: 18
```

5. Ext.air.FileTreeLoader

这个类可以把本地的目录和文件结构显示到树中，如图C-27所示。

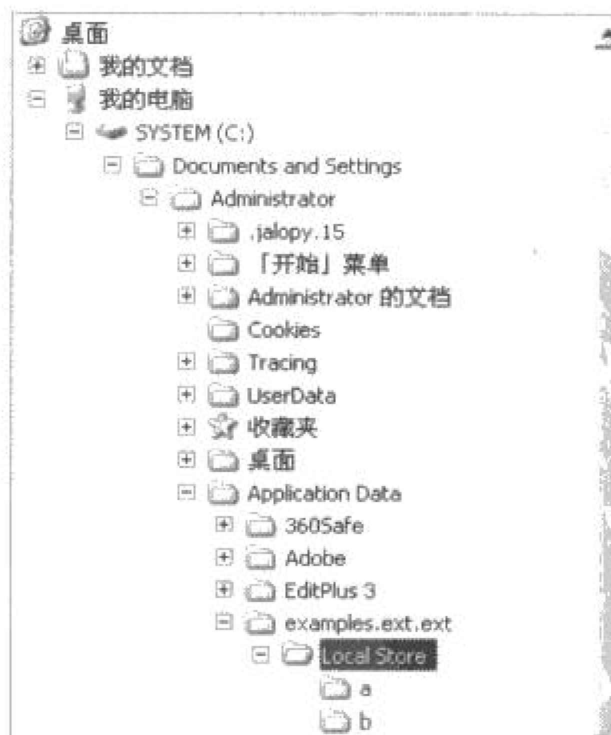


图C-27　使用FileTreeLoader读取的目录树

实例代码如下：

```
var tree = new Ext.tree.TreePanel({
    autoHeight: true,
    loader: new Ext.air.FileTreeLoader({
        directory: air.File.applicationStorageDirectory.resolvePath('.')
    }),
    root: new Ext.tree.AsyncTreeNode({
        text: 'root'
    })
});
tree.render('tree');
tree.expand();
```

除了使用了`Ext.air.FileTreeLoader`之外，其他的部分都与之前树形的配置相同，在创建`Ext.air.FileTreeLoader`的时候还要指定根目录，我们这里使用`air.File.applicationStorageDirectory.resolvePath('.')`来获得当前应用在系统中的存储目录，如图C-28所示。通常是在C:\Documents and Settings\Administrator\Application Data\examples.ext.ext\Local Store下，手工在这里建了几个目录和文件，之后AIR就可以递归目录结构生成树形效果了。



图C-28　FileTreeLoader默认读取的目录位置

6. Ext.air.MusicPlayer

在EXT 2.0.2中我们提过的`Ext.air.Sound`类，主要适用于短暂的声音，如蜂鸣和闹钟。而

Ext.air.MusicPlayer可以用于播放较长的声音，如音乐、歌曲等。MusicPlayer支持所有对于音乐的基本操作，包括暂停、停止、播放或跳转等。使用MusicPlayer的基本代码如下：

```
var mp = new Ext.air.MusicPlayer();
mp.adjustVolume(0.5);
mp.play('beep.mp3');
```

7. **Ext.air.Notify**

我们使用这个类，可以实现MSN那种提示信息，就是从屏幕的右下角突然弹出一个带有提示内容的小窗口。实例代码如下：

```
var notify = new Ext.air.Notify({
    extAllCSS: 'ext-2.2.1/resources/css/ext-all.css',
    msg: 'Message',
    icon: 'ext-air/resources/icons/extlogo128.png'
});
```

不过在测试的时候，因为是直接使用adl.exe执行application.xml，CSS样式和图片都显示不出来，估计是需要将应用打包以后才能使用相对路径中的CSS样式和图片。

8. **Ext.air.VideoPanel**

最后这个是支持Flash视频的面板，它还注册了videopanel这个xtype，我们可以把它直接应用在布局中。

有两种方式可以在VideoPanel中播放视频，一种是传入视频的url，代码如下：

```
var viewport = new Ext.Viewport({
    layout: 'fit',
    items:[{
        id: 'video',
        url: 'http://www.mediacollege.com/video-gallery/testclips/barsandtone.flv',
        xtype: 'videopanel'
    }]
});
```

另一种方法是调用loadVideo()函数，代码如下：

```
Ext.getCmp('video').loadVideo('http://www.mediacollege.com/video-gallery/
testclips/barsandtone.flv');
```

不过使用loadVideo()之前，要保证VideoPanel已经渲染完成了，否则就会报错。